APPLICATION OF MARQUARDT'S NONLINEAR LEAST SQUARES ALGORITHM TO FREE-FLIGHT YAW DATA ANALYSIS

James W. Bradley

Ballistic Research Laboratories
Aberdeen Proving Ground, Maryland

September 1975

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

BRL

MEMORANDUM REPORT NO. 2526

APPLICATION OF MARQUARDT'S NONLINEAR LEAST SQUARES ALGORITHM TO FREE-FLIGHT YAW DATA ANALYSIS

James W. Bradley

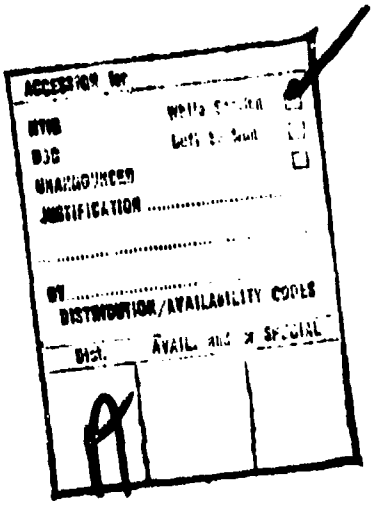
September 1975

USA BALLISTIC RESEARCH LABORATORIES
ABERDEEN PROVING GROUND, MARYLAND

Destroy this report when it is no longer needed.
Do not return it to the originator.

Secondary distribution of this report by originating or sponsoring activity is prohibited.

Additional copies of this report may be obtained from the National Technical Information Service, U.S. Department of Commerce, Springfield, Virginia 22151.



The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: BRL Memorandum Report No. 2526	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): APPLICATION OF MARQUARDT'S NONLINEAR LEAST SQUARES ALGORITHM TO FREE-FLIGHT YAW DATA ANALYSIS	5. TYPE OF REPORT & PERIOD COVERED: Final / 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): James W. Bradley	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: USA Ballistic Research Laboratories, Aberdeen Proving Ground, Maryland 21005	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: RDT&E 1T061102A33D
11. CONTROLLING OFFICE NAME AND ADDRESS: U.S. Army Materiel Command, 5001 Eisenhower Avenue, Alexandria, Virginia 22333	12. REPORT DATE: SEPTEMBER 1975 / 13. NUMBER OF PAGES: 54
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified / 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Curve Fitting
Marquardt Algorithm
Nonlinear Least Squares
Parameter Optimization
Yaw Reduction

20. ABSTRACT (Continue on reverse side if necessary and identify by block number) ds

Marquardt's algorithm is an improved technique for the least squares estimation of nonlinear parameters. The present report describes the algorithm and documents a FORTRAN subroutine for carrying it out. A specific application of the subroutine is discussed in detail: fitting a complex yaw equation to data obtained from a Free Flight Range. An example is given in which the fitting process converged to the wrong answer without the algorithm and to the right answer with it.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

I. INTRODUCTION

Fitting an equation to a mass of data points (x, y) is an ancient and honorable pastime. Circa 1800, Gauss and Legendre discovered independently what proved to be a giant advancement in technique: the Method of Least Squares. Refinements of this basic method are still showing up in the literature. This report concerns one such refinement, proposed by D. W. Marquardt[1].

Suppose we are given an equation of the form

$$y = f(x, P) \tag{1}$$

where P is a set of n unknown constant parameters:

$$P = \{a_1, a_2, \ldots a_n\} \tag{2}$$

We are also given a set of m measurements

$$Y = \{y_1, y_2, \ldots y_m\} \tag{3}$$

at the corresponding (presumably error-free) x values $x_1, x_2, \ldots x_m$, where m exceeds n: there are more measurements than unknown parameters. For convenience, we assume each measurement y_i is equally reliable; this eliminates the need for weighting factors, which add nothing to the concepts involved.

We are asked to determine the least squares fit of Equation (1) to the set of measurements. That is, we seek those values of the a_j's that minimize ε, the sum of the squares of the residuals of the fit:

$$\varepsilon = F(P) \equiv \sum_{i=1}^{m} [y_i - f(x_i, P)]^2 \tag{4}$$

The criterion function F in (4) is assumed to be an analytic function of the n parameters. We can describe the situation geometrically as follows. Consider an n-dimensional Euclidean space S in which any set of parameter values constitutes the coordinates of a point P. Then

---

1. *Donald W. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," J. Soc. Indust. Appl. Math., Vol. 11, No. 2, pp 431-441, June 1963.*

ε is the value of the continuous scalar point function F at point P. For each point P in the parameter space S there corresponds a single value of ε and our task is to search S in some well-defined, automated manner for the point (and hopefully there is only one such point) that yields the absolute minimum value.

In practice, the criterion function often possesses one or more local minimum values in addition to the desired absolute minimum. The particular minimum we reach then depends on where in space S we start our search. If we start too close to the point L associated with one of these local minima, and if our mathematical searching procedure — in the interest of computer economy — is not prohibitively sophisticated, then we are going to be drawn irresistibly to L. Even if we start far from L, a funny thing can happen on the way to the right answer: our path may pass through L's sphere of influence and we are trapped. We will give an example of this misfortune in the final section of this report.

A necessary (but insufficient) condition for ε to be at a local or absolute minimum is that the gradient of ε be the zero vector:

$$\left(\frac{\partial \varepsilon}{\partial a_1}, \frac{\partial \varepsilon}{\partial a_2}, \ldots \frac{\partial \varepsilon}{\partial a_n} \right)_S = \vec{0} \qquad (5)$$

Thus we seek the set of parameter values that satisfy all n component equations of (5) simultaneously.

When Equation (1) is linear in the parameters:

$$y = a_1 \phi_1 (x) + a_2 \phi_2 (x) + \ldots + a_n \phi_n (x) \qquad (6)$$

Equation (5) represents a system also linear in the parameters. In principle, such a system (provided it is nonsingular and stable) can be solved by any one of a variety of well-known techniques. This is too easy; thus we specify here that Equation (1) is nonlinear in at least one of the parameters.

For this nonlinear situation, two of the estal ished methods for minimizing ε are:

(a) steepest descent (alias gradient search);

(b) differential corrections (alias Taylor series expansion, alias Gauss-Newton method).

Marquardt has proposed a third technique. As we will see, his algorithm[1] represents a blend of the first two methods, retaining the best features of each.

In Sections II to IV, we will discuss enough of the differential corrections and steepest descent techniques to see what is involved in an interpolation of the two. In Sections V to VII, we will discuss the Marquardt algorithm and its implementation as a FORTRAN subroutine MARQ. In Sections VIII to XI, we will show how subroutine MARQ is used to fit a complex yaw equation to a set of measurements obtained from an enclosed free-flight spark-photography range. In the final section we will compare a case run with and without the aid of Marquardt's algorithm.

II. DIFFERENTIAL CORRECTIONS

We can obtain the basic equations of the differential corrections technique by approximating y or ε or grad ε by a first-order Taylor expansion about a given point P_o. For our purposes, the simplest choice is grad ε:

$$\frac{\partial \varepsilon}{\partial a_k} \cong \left[\frac{\partial \varepsilon}{\partial a_k}\right]_o + \sum_{j=1}^{n} \left[\frac{\partial^2 \varepsilon}{\partial a_j \, \partial a_k}\right]_o \Delta a_j \qquad (7)$$

where

Δa_j = the change in a_j in moving from P_o to P

and where the zero subscript denotes evaluation at point P_o. Under approximation (7), the nonlinear system (5) is replaced by a system that is linear in the parameter increments Δa_j:

$$\sum_{j=1}^{n} \left[\frac{\partial^2 \varepsilon}{\partial a_j \, \partial a_k}\right]_o \Delta a_j = -\left[\frac{\partial \varepsilon}{\partial a_k}\right]_o, \; k = 1, 2, \ldots n \qquad (8)$$

or, simplifying the notation,

$$\sum_{j=1}^{n} \alpha_{jk} \, \Delta a_j = \beta_k, \; k = 1, 2, \ldots n \qquad (9)$$

where

$$\beta_k = -\frac{1}{2}\left[\frac{\partial \varepsilon}{\partial a_k}\right]_o \qquad (10)$$

$$\alpha_{jk} = \frac{1}{2}\left[\frac{\partial^2 \varepsilon}{\partial a_j \, \partial a_k}\right]_o \tag{11}$$

and where the factor 1/2 has been introduced to simplify later expressions for β_k and α_{jk}.

The system (9) can be written in matrix form as

$$\vec{\Delta P} \cdot \alpha = \vec{\beta} \tag{12}$$

where

α = the n x n symmetric matrix $\left(\alpha_{jk}\right)_S$

$\vec{\Delta P}$ = the row* vector $\left(\Delta a_1, \Delta a_2, \ldots \Delta a_n\right)_S$

$\vec{\beta}$ = the row* vector $\left(\beta_1, \beta_2, \ldots \beta_n\right)_S$

The symmetric matrix α is sometimes called the curvature matrix because it is a measure of the curvature of the ε hypersurface in the parameter space. From (10), we see that $\vec{\beta}$ is a vector in the direction of the negative gradient of ε at P_o:

$$\vec{\beta} = -\frac{1}{2}\,(\text{grad } \varepsilon)_o \tag{13}$$

Equation (9), or the equivalent Equation (12), represents a system of n linear equations in the n increments Δa_j. Hence we can solve the system for these increments:

$$\Delta a_j = \sum_{k=1}^{n} \beta_k \, \tilde{\alpha}_{jk}, \quad j = 1, 2, \ldots n \tag{14a}$$

or

$$\vec{\Delta P} = \vec{\beta} \cdot \tilde{\alpha} \tag{14b}$$

If we had written the product in Eq. (12) as $\alpha \cdot \vec{\Delta P}$, then $\vec{\Delta P}$ and $\vec{\beta}$ would be column vectors.

where

$$\tilde{\alpha} = \left(\tilde{\alpha}_{jk}\right)_S$$

= the inverse of matrix α

$$\tilde{\alpha}_{jk} = \frac{\text{cofactor of element } \alpha_{jk}}{\text{determinant of } \alpha}$$

Of course, to solve the system (9), we must be able to express β_k and α_{jk} in terms of the given fitting function of Eq. (1). Substituting Eq. (4) in (10), we have:

$$\beta_k = -\frac{1}{2}\left\{-2\sum_{i=1}^{m}\left[y_i - f\left(x_i, P_o\right)\right]\left[\frac{\partial f\left(x_i, P\right)}{\partial a_k}\right]_o\right\}$$

$$= \sum_{i=1}^{m}\left[y_i - f\left(x_i, P_o\right)\right] D_{ik} \tag{15}$$

where

$$D_{ik} = \left[\frac{\partial f\left(x_i, P\right)}{\partial a_k}\right]_o \tag{16}$$

Similarly, substituting (4) in (11), we have

$$\alpha_{jk} = \sum_{i=1}^{m}\left\{D_{ij}\, D_{ik} - \left[y_i - f\left(x_i, P_o\right)\right]\left[\frac{\partial^2 f\left(x_i, P\right)}{\partial a_j\, \partial a_k}\right]_o\right\} \tag{17}$$

The second-order term in (17) could be a big nuisance if we let it, but traditionally — and with ample justification — this term is dropped. Hence our working definition of α_{jk} is the approximation

$$\alpha_{jk} \cong \sum_{i=1}^{m} D_{ij}\, D_{ik} \tag{18}$$

When the system (12) has been solved for the increment vector, we can then make the differential corrections on P_o, that is, we can then obtain a new point P_1 whose vector is

$$\vec{P}_1 = \vec{P}_o + \vec{\Delta P} \tag{19}$$

and whose associated error value is $\varepsilon_1 = F(P_1)$.

If Eq. (7) were exact (which would happen only if the original fitting function in Eq. (1) were linear in the parameters a_j), then P_1 would be the desired point at which ε is a minimum. Since Eq. (7) is only an approximation, P_1 is <u>not</u> the solution to the nonlinear problem. Indeed, if the starting point P_o is not close enough to the solution point (and it is usually difficult to say beforehand how close is "close enough"), then ε_1 will be larger than ε_o; the process may or may not recover from this inauspicious start. By the rules of the game, P_1 is made the new starting point, whether or not it is an improvement. System (12) is then re-solved for a new set of parameter increments, which in turn yield — for better or for worse — a new point P_2 and so on, each iteration of the scheme taking us to the next point. This process <u>may</u> converge to a unique point P_M and if so, $\varepsilon_M \equiv F(P_M)$ <u>may</u> be the absolute minimum value of the criterion function. But don't count on it.

An estimate of the error of the fit at any point P is usually obtained by the relation

$$E = b\sqrt{\frac{\varepsilon}{m - n}} \tag{20}$$

where

b = 1.0 to obtain an estimate of the root-mean-square error (more precisely, the RMS deviation between the measurements and a hypothetical set of "true" measurements free from observational errors)

= 0.67449 to obtain an estimate of the probable error

and an estimate of the error in any parameter a_j at point P is given by

$$E_j = E\sqrt{\alpha_{jj}} \tag{21}$$

Because of its application in Equation (21), the inverse matrix $\tilde{\alpha}$ is sometimes called the error matrix.

Before we consider the steepest descent approach, it will be convenient to simplify whatever physical dimensions are involved in our equations.

III. SCALING

If the parameters a_j are dimensional and — worse yet — not all of the same dimensions, then our parameter space S can be a hodge-podge of femto-drams, fermis and fortnights. Letting $[\ ]_d$ denote "dimensions of", we have

$$\left.\begin{aligned} \left[D_{ik}\right]_d &= \left[y/a_k\right]_d \\ \left[\alpha_{jk}\right]_d &= \left[y^2/a_j\ a_k\right]_d \\ \left[\frac{\partial \epsilon}{\partial a_k}\right]_d &= \left[\beta_k\right]_d = \left[y^2/a_k\right]_d \end{aligned}\right\} \tag{22}$$

Suppose now that we introduce a scaled parameter space $\hat{S}$ in which

$$\hat{a}_k = a_k \sqrt{\alpha_{kk}} \tag{23}$$

so that

$$\left[\hat{a}_k\right]_d = [y]_d,\ k = 1, 2, \ldots n \tag{24}$$

Then the correspondingly scaled partial derivatives and $\hat{\alpha}$, $\hat{\beta}$ and grad ϵ elements are, by Equations (16), (18) and (15):

$$\left.\begin{aligned} \hat{D}_{ik} &= D_{ik}/\sqrt{\alpha_{kk}} \\ \hat{\alpha}_{jk} &= \alpha_{jk}/\sqrt{\alpha_{jj}\ \alpha_{kk}} \\ \hat{\beta}_k &= \beta_k/\sqrt{\alpha_{kk}} \\ \frac{\partial \epsilon}{\partial \hat{a}_k} &= \left[\frac{\partial \epsilon}{\partial a_k}\right]/\sqrt{\alpha_{kk}} \end{aligned}\right\} \tag{25}$$

with dimensions

$$\left.\begin{aligned} \left[\hat{D}_{ik}\right]_d &= \left[\hat{\alpha}_{jk}\right]_d = 1 \\ \left[\frac{\partial \varepsilon}{\partial \hat{a}_k}\right]_d &= \left[\hat{\beta}_k\right]_d = [y]_d \end{aligned}\right\} \tag{26}$$

Moreover, Cauchy's inequality applied to definition (18) asserts that

$$\alpha_{jk}^2 \leq \alpha_{jj}\, \alpha_{kk}$$

so that

$$-1 \leq \hat{\alpha}_{jk} \leq 1 \tag{27}$$

Thus we have transformed into a space $\hat{S}$ in which:

(a) each parameter and each component of the vectors grad ε and $\vec{\beta}$ has the same dimension, namely $[y]_d$;

(b) $\hat{\alpha}$ is a dimensionless matrix whose diagonal elements are all unity and whose off-diagonal elements satisfy Inequality (27).

All equations of the previous section hold for the transformed elements; in particular, Equation (9) becomes

$$\sum_{j=1}^{n} \hat{\alpha}_{jk} \cdot \Delta\hat{a}_j = \hat{\beta}_k, \quad k = 1, 2, \ldots n \tag{28}$$

with solution

$$\Delta\hat{a}_j = \sum_{k=1}^{n} \hat{\beta}_k \tilde{\hat{\alpha}}_{jk}, \quad j = 1, 2, \ldots n \tag{29}$$

Transformation (23) is often used in linear least-squares to improve the numerical behavior of the computations: in particular, inversion of the curvature matrix. For our nonlinear, iterative problem, the transformation could have the same salutary effect but at the cost of a little more work. This is because the values of the scale factors

$\sqrt{\alpha_{kk}}$ depend on the current set of parameter values. Hence each time the parameters are up-dated, a new transformation must be made: a new $\hat{S}$ space created. This is not a big problem.

The transformation to $\hat{S}$ is not essential in the differential corrections method; it is needed more in the steepest descent approach. Some of the properties of the latter technique are not scale-invariant, so that choosing the right space can be important. At the very least, it is convenient to proceed by steepest descent in a space in which the components of grad ϵ are dimensionally equal.

IV. STEEPEST DESCENT

Provided that $|\text{grad } \epsilon|$ is not zero at the current point P_o (if it were, P_o would be the desired solution), then -grad ϵ at that point is a vector in whose direction ϵ will decrease most rapidly (at least at first) as we move away from P_o. If P_g is any other point along this negative gradient, we have from Eq. (13):

$$\vec{P}_g\,(h) = \vec{P}_o + h\,\vec{\beta} \tag{30}$$

where h is a dimensionless positive constant and where the three vectors are to be resolved into components in the scaled space $\hat{S}$. Thus in component form, (30) becomes:

$$\Delta\hat{a}_k = h\,\hat{\beta}_k \tag{31}$$

In the steepest descent technique, we choose that h for which ϵ is a local minimum along the gradient. Perhaps the simplest way to do this is to sample ϵ at appropriately small intervals as we move away from P_o along the negative gradient. As soon as we reach a point where ϵ has increased, we interpolate between that point and the previous sampling point to determine where ϵ has a local minimum. Then we evaluate the negative gradient at this new point and start off again in the new direction.

The difficulty with this approach is that in the neighborhood of the solution point, where $|\text{grad } \epsilon|$ is nearly zero, further progress is painfully slow. Often, the sampling size must be shortened beyond all endurance. Ingenious variations on the basic steepest descent theme have lessened this difficulty but not removed it.

V. INTERPOLATION

Comparing the two methods just discussed, we note that:

(a) Far from the solution point, the steepest descent technique is superior. It must proceed so as to decrease ε, whereas the differential corrections method has no such obligation and is likely to diverge.

(b) Close to the solution point, the differential corrections method is superior. It converges in the very region where the steepest descent technique languishes.

Marquardt[1] has proposed an interpolation between the two methods: a technique that behaves like the steepest descent when we are far from the solution and like the differential corrections method when we enter the neighborhood in which the linear truncation, Equation (7), is adequate.

To achieve this interpolation, a positive, non-dimensional constant λ is added to each diagonal element of the transformed curvature matrix. That is, Equation (28) is replaced by

$$\sum_{j=1}^{n} \gamma_{jk} \cdot \Delta\hat{a}_j = \hat{\beta}_k, \quad k = 1, 2, \ldots n \tag{32}$$

where

$$\gamma_{jk} = \begin{cases} 1 + \lambda & \text{when } j = k \\ \hat{\alpha}_{jk} & \text{when } j \neq k \end{cases} \tag{33}$$

We see that:

(a) As $\lambda \to \infty$, the diagonal terms of the system dominate and Equation (32) degenerates into n uncoupled equations of the form

$$(1 + \lambda)\, \Delta\hat{a}_k = \hat{\beta}_k$$

or, since $\lambda >> 1$ by assumption,

$$\Delta\hat{a}_k = \hat{\beta}_k/\lambda \tag{34}$$

Comparing this result with Equation (31), we see that for very large λ, Equation (32) simulates the steepest descent approach with $h = 1/\lambda$. (As we will see, however, λ is not determined by the same criterion — a locally minimum ε — used to obtain h.)

(b) As $\lambda \to 0$, Equation (32) approaches Equation (28), that is, Equation (32) reduces to the differential corrections method.

The interpolation sends us from a given point P_o in the direction of the vector

$$\vec{\Delta P} = \left(\Delta\hat{a}_1, \Delta\hat{a}_2, \ldots \Delta\hat{a}_n\right)_S$$

satisfying (32), whereas the gradient method sends us in the direction of the vector $\vec{\beta}$. The angle between these two directions:

$$\theta = \cos^{-1}\left(\frac{\vec{\beta} \cdot \vec{\Delta P}}{|\vec{\beta}| \, |\vec{\Delta P}|}\right) \tag{35}$$

always lies between 0 and 90°. (This angle is the same, of course, in S and $\bar{S}$.) Marquardt proves that for the given point P_o (and hence for a given $\vec{\beta}$), θ is a continuous, monotonically decreasing function of λ, such that:

(a) As $\lambda \to \infty$, $\theta \to 0$.

(b) As $\lambda \to 0$, θ approaches some value θ_{MAX} less than 90°. (Marquardt states that θ_{MAX} usually lies between 80° and 90°, indicating that the differential corrections method usually proceeds almost at right angles to the gradient approach. However, in the yaw fit application to be discussed later, I have found that θ_{MAX} is usually less than 80° and sometimes less than 30°.)

The insertion of λ into (28) makes interpolation possible; an effective interpolation is achieved only when we have a sound strategy for changing the value of λ according to whether ε has increased or decreased since its previous evaluation. In the next section, we present the strategy proposed by Marquardt.

VI. MARQUARDT'S ALGORITHM

Let $\varepsilon_o = F(P_o)$ be the value of the criterion function at some current point P_o. For any λ, we can solve (32) for the increments $\vec{\Delta P}$, obtain $\vec{P}_1 = \vec{P}_o + \vec{\Delta P}$ and evaluate

$$\varepsilon_1 = F(P_1) = G(P_o, \lambda)$$

where the second functional form emphasizes the dependence of ε_1 on both the previous point P_o and the current value of λ.

In the Marquardt approach, the point P_1 is only a candidate for the point that will replace P_o as our current set of parameter values. A new rule is this: we won't move from P_o to any new point P_1 unless ε is smaller at the new point than at P_o:

$$\varepsilon_1 < \varepsilon_o \tag{36}$$

For any point P_o that is not already a local minimum, there always exists (at least in theory) a λ_M such that for $\lambda > \lambda_M$, (36) holds. The question is: how much larger than λ_M should we choose our λ? Marquardt's answer: no larger than necessary within a factor of $A > 1$. That is, if our initial λ doesn't work, try $A\lambda$, $A^2\lambda$, $A^3\lambda$, etc., quitting as soon as (36) holds. Ten is the suggested value for A.

Of course, we could determine λ so as to minimize ε locally, just as with h in the steepest descent approach. Marquardt points out, however, that this would usually result in a much larger λ than needed to satisfy (36); we would be faced with all the disadvantages of the steepest descent.

On the other hand, we would like to simulate the differential corrections method as soon as we get within the radius of convergence of that technique. Hence, when things are going well, we want to reduce λ, say by factors of A again. Marquardt suggests, however, that if λ is already negligible compared with 1 to the number of significant digits carried, then there is no need to reduce λ further.

Marquardt's strategy can then be summarized by the flow-chart of Figure 1. (This is, at any rate, my understanding of Marquardt's strategy; his presentation[1] differs considerably in manner and form.)

Note from the figure that in practice Marquardt replaces (36) by the condition

$$\varepsilon_1 \leq \varepsilon_o \tag{36a}$$

The distinction may seem academic but it's not. In an early version of our program for implementing the algorithm, I unwisely used (36) instead of (36a). Why, after all, should we move to P_1 if the error there is the same as at P_o? I very soon found out why and the answer (upon after-

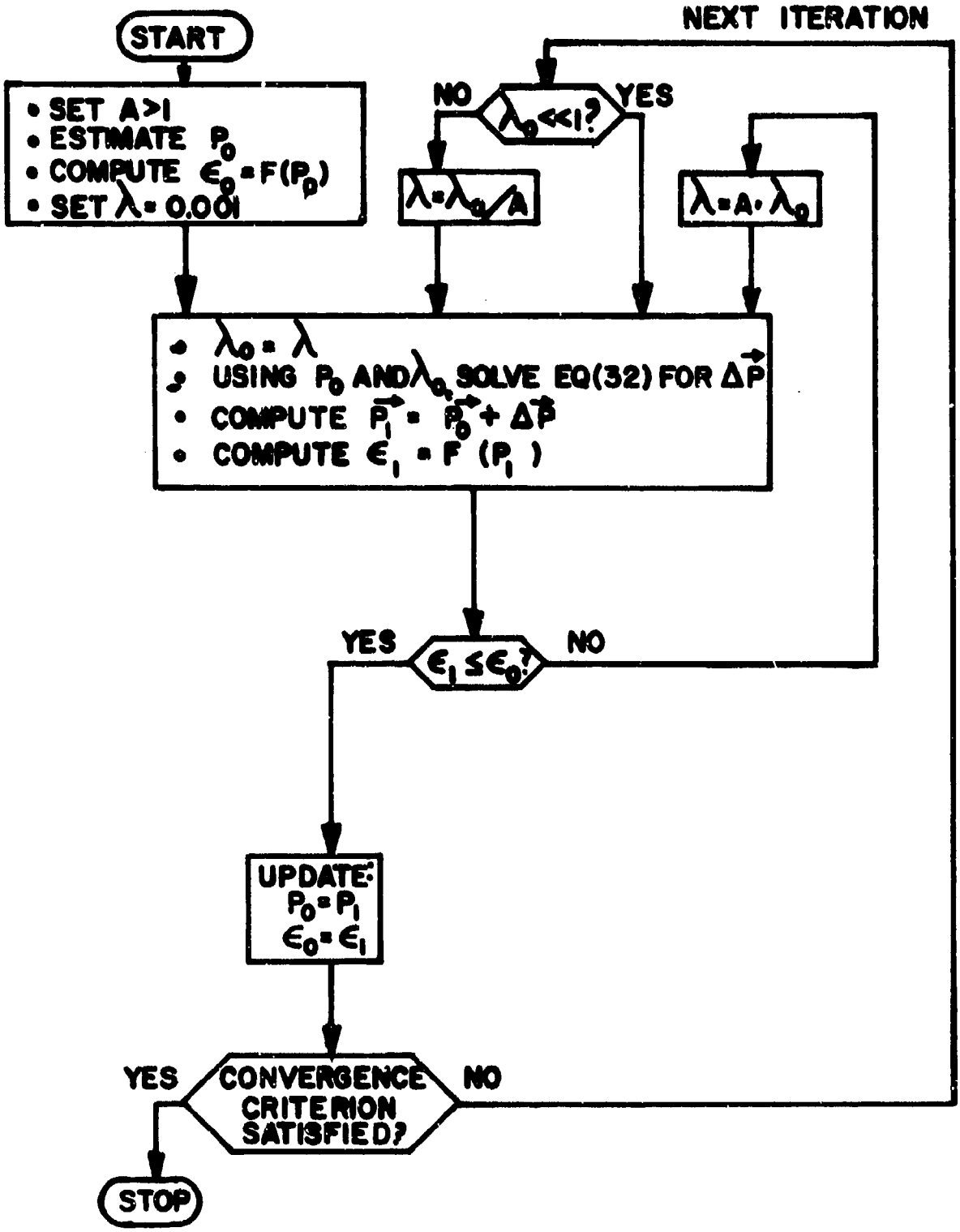


FIG. 1 MARQUARDT ALGORITHM

thought) is obvious. It is possible for the current point P_o at the end of some iteration to "be" the solution point in the sense that the machine will be unable to find a point with a smaller associated error (to the number of digits carried). In this situation, of course, we should exit the loop triumphantly and proceed to analyze results. The trouble is: any convergence test for determining whether or not to start another iteration is necessarily imperfect. The test is usually based solely on how much the current point P_o and/or error ε_o differs from the values of the previous iteration. Thus we could close in on the answer so quickly that the solution point differs from the previous point by more than our criterion allows. In this event, another iteration will begin: a hopeless search for a candidate P_1 satisfying (36).

As an example of what can happen, consider the test case whose results are summarized in Table 1. The fitting equation here was the complex yaw equation, to be discussed later; the convergence criterion was that $(1 - \varepsilon_1/\varepsilon_o)$ be less than 0.00001. In the first iteration, the process obtained an ε_1 (.005307) less than ε_o (.007418), so that it would make no difference whether condition (36) or (36a) was used in the scheme of Figure 1. The convergence condition was not satisfied (since $1 - \varepsilon_1/\varepsilon_o = 0.285$) so the second iteration was began. Similarly, the third and fourth iterations were begun. It was only in attempting to complete the fourth iteration that the superiority of (36a) to (36) became evident.

The condition (36a), $\varepsilon_1 \leq \varepsilon_o$, was finally met (in this case, when ε_1 achieved equality with ε_o to the sixteen decimal digits carried*) at $\lambda = 10^{13}$. When condition (36), $\varepsilon_1 < \varepsilon_o$, was substituted for (36a), the fourth iteration was never completed; program execution was interrupted at $\lambda = 10^{16}$ by a floating point overflow. The hang-up probably occurred in evaluating the determinant of matrix γ, Eq. (33). When λ is large, say $\lambda = 10^k$ where $k \geq 2$, then the determinant is approximately the product of the diagonal elements of γ:

$$\text{DET} \cong (1 + \lambda)^n \cong 10^{kn}$$

If a computer can't handle real numbers larger than 10^j, then we are in trouble when k exceeds j/n. For our computer, j = 154 and for the test case, n = 10; hence the hang-up at k = 16.

---

*Of course, it is unlikely that either ε_o or ε_1 is correct to the sixteenth digit; the vagaries of round-off error alone may make the last few digits meaningless.

Table 1. Test Case Using the Marquardt Algorithm of Figure 1 with A = 10

Iteration	$\log_{10} \lambda$	$10^3 \lvert\vec{\beta}\rvert$	$10^3 \lvert\vec{\Delta P}\rvert$	θ (deg)	ε
0					.007418
1	- 3	19.8	49.6	50.80	.005307
2	- 4	2.8	5.6	43.23	.005274
3	- 5	.45	1.2	29.57	.005269, 809717, 322701
4(1)	- 6	.13564	.404162	34.5383	.005270, 482320
(2)	- 5	.13564	.404146	34.5376	.005270, 482291
(3)	- 4	.13564	.40398	34.5306	.005270, 481995
(4)	- 3	.13564	.40234	34.4612	.005270, 479055
(5)	- 2	.13564	.38679	33.8029	.005270, 451141
(6)	- 1	.13564	.28540	29.4097	.005270, 265439
(7)	0	.13564	.08905	17.2026	.005269, 922121
(8)	1	.13564	$.12611 \times 10^{-1}$	3.8898	.005269, 820013
(9)	2	.13564	$.13453 \times 10^{-2}$	.4496	.005269, 810634
(10)	3	.13564	$.13552 \times 10^{-3}$	.0457	.005269, 809807
(11)	4	.13564	$.13563 \times 10^{-4}$	.0046	.005269, 809726
(12)	5	.13564	$.13564 \times 10^{-5}$	.0005	.005269, 809718
(13)	6	.13564	$.13564 \times 10^{-6}$	.0000	.005269, 809717, 412691
(14)	7	.13564	$.13564 \times 10^{-7}$	.0000	.005269, 809717, 331701
(15)	8	.13564	$.13564 \times 10^{-8}$	.0000	.005269, 809717, 323603
(16)	9	.13564	$.13564 \times 10^{-9}$	.0000	.005269, 809717, 322794
(17)	10	.13564	$.13564 \times 10^{-10}$	.0000	.005269, 809717, 322711
(18)	11	.13564	$.13564 \times 10^{-11}$	.0000	.005269, 809717, 322703
(19)	12	.13564	$.13564 \times 10^{-12}$	.0000	.005269, 809717, 322702
4(20)	13	.13564	$.13564 \times 10^{-13}$	.0000	.005269, 809717, 322701

In our limited experience, the test case of Table 1 is not typical; we actually contrived this unimpressive example. Still, it could have happened in "real life", so let us try to profit by it.

We see from Table 1 that a lot of needless work was done in the fourth iteration — the matrix equation (32) was solved twenty times — only to finish at the point already obtained in the third iteration. Clearly, there is room here for improvement in technique.

Of the various ways of reducing the number of calculations, the simplest might to be to introduce <u>two</u> acceptable conditions:

$$\varepsilon_1 < \varepsilon_0 \tag{36}$$

and

$$\varepsilon_0 < \varepsilon_1 \leq \varepsilon_0 (1 + \Delta) \tag{36b}$$

where Δ is assigned some small positive value. If condition (36) is satisfied, we up-date the current point and its associated error and proceed as in Figure 1. If, instead, condition (36b) is satisfied, we <u>assume</u> that the point reached at the conclusion of the previous iteration is as good as we are going to get. In this event, the entire process ends: that previous point and its associated error are taken to be the final point and error. For example, if we take $\Delta = 0.00001$, condition (36b) would be satisfied at the end of the eighth loop — iteration 4(8) — of Table 1. At this stage, the program would assume that the solution point is the point obtained by the third iteration.

The above technique may be an excellent <u>practical</u> way out of our difficulties; yet there is something vaguely <u>unpleasant</u> about condition (36b). In effect, we are giving up. The suspicion would always linger that in another loop or two within the iteration, the situation might suddenly improve and condition (36) would be satisfied.

We ask then: how can we cut down on the computations without relaxing our standards; that is, how can we satisfy (36a) with less effort? One obvious answer is suggested by Eq. (34):

$$\overrightarrow{\Delta P} = \vec{\beta}/\lambda, \quad \lambda >> 1 \tag{34a}$$

a relationship confirmed by Table 1. For large values of λ, it is no longer necessary to obtain new $\overrightarrow{\Delta P}$ candidates by the time-consuming process of increasing λ and re-solving Eq. (32) by matrix inversion. All succeeding $\overrightarrow{\Delta P}$ candidates within that iteration will be very nearly parallel and hence can be obtained directly by a scale change:

$$(\overrightarrow{\Delta P})_{\substack{\text{next} \\ \text{candidate}}} = (\overrightarrow{\Delta P})_{\substack{\text{rejected} \\ \text{candidate}}} \Big/ B, \quad B > 1 \tag{37}$$

where B need not have the same value assigned to A.

We now ask: is it necessary to wait until λ is large to replace (32) by (37)? Marquardt mentions that in some circumstances (characterized by matrix elements $\hat{\alpha}_{jk}$ exceeding 0.99) he found it helpful to revert to Eq. (37) as soon as the angle θ fell below, say 45° (that is, when he was within 45° of the steepest descent direction). Usually open to a good suggestion, I incorporated this little sub-scheme in the grand plan as a feature operative under all circumstances. That is, whenever θ is less than some specified value G and condition (36a) isn't satisfied for vector $\vec{P} + \Delta\vec{P}$, the next vector considered is $\vec{P} + \Delta\vec{P}/B$.

The final form of our modified Marquardt algorithm is shown in Figure 2. If the user feels that the $\theta < G$ feature is too great a violation of the spirit of Marquardt's algorithm, he need only set G at zero to avoid that feature.

Table 2 shows the results of applying the modified algorithm, with A = B = 10, G = 45°, to the test case of Table 1. The first three iterations are identical, but the fourth has been improved considerably. The reduction in the number of loops in the fourth iteration is not as important as the fact that each loop required significantly fewer machine calculations. Of course, we could introduce condition (36b) into this new scheme as well. With $\Delta = 0.00001$, the process would then end at iteration 4(2).

After we have discussed our yaw equation, we will present another test case in more detail. First, however, we introduce the FORTRAN subroutine MARQ for carrying out the algorithm of Figure 2.

VII. COMMENTS ON SUBROUTINE MARQ

At least five programs to implement Marquardt's algorithm pre-date the FORTRAN subroutine MARQ listed in Appendix A:

(a) A FORTRAN program "Least-Squares Estimation of Nonlinear Parameters" cited in Reference 1 and available as IBM Share Program No. 1428.

(b) A program "Non-Linear Least Squares (GAUSHAUS)," University of Wisconsin Computing Center, written by D. A. Meeter in 1964.

(c) A "Nonlinear Least-Squares Curve Fitting Program," a 1966 modification of (b) above by F. S. Wood. A User's Manual but no listing of this program is given in Reference 2; the program, a FORTRAN

---

2. *Cuthbert Daniel and Fred S. Wood, Fitting Equations to Data: Computer Analysis of Multifactor Data for Scientists and Engineers, Wiley-Interscience, New York, 1971.*

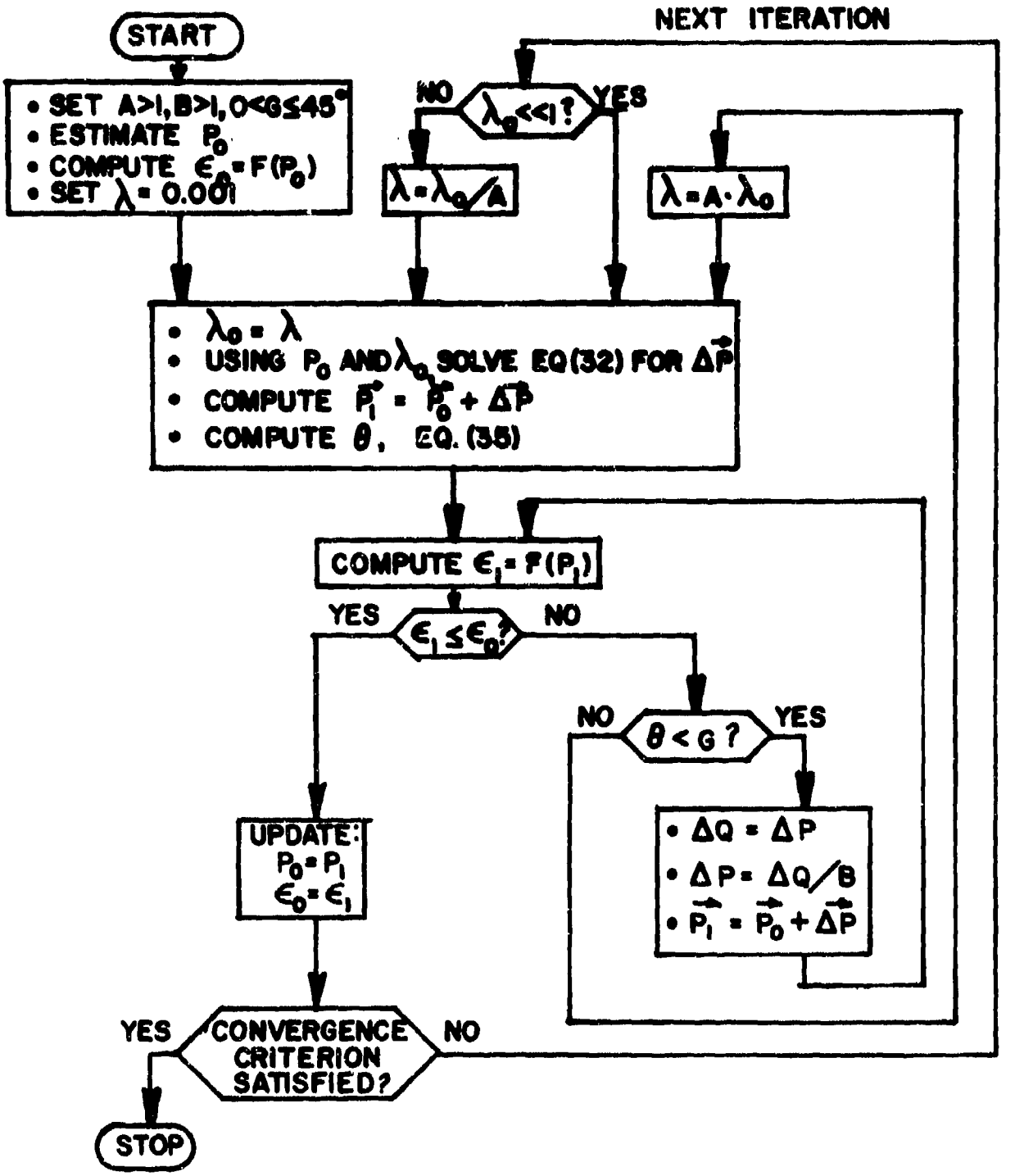


FIG. 2 MODIFIED MARQUARDT ALGORITHM

Table 2. Same Test Case as in Table 1, but Using the Modified Marquardt Algorithm of Figure 2 with A = B = 10, G = 45°

Iteration	$\log_{10} \lambda$	$10^3 \lvert\vec{\beta}\rvert$	$10^3 \lvert\vec{\Delta P}\rvert$	θ (deg)	ε
0					.007418
1	- 3	19.8	49.6	50.80	.005307
2	- 4	2.8	5.6	43.27	.005274
3	- 5	.45	1.2	29.57	.005269, 809717, 322701
4(1)	- 6	.13564	.404162	34.54	.005270, 482320
(2)	- 6	.13564	$.404162 \times 10^{-1}$	34.54	.005269, 852083
(3)	- 6	.13564	$.404162 \times 10^{-2}$	34.54	.005269, 813705
(4)	- 6	.13564	$.404162 \times 10^{-3}$	34.54	.005269, 810114
(5)	- 6	.13564	$.404162 \times 10^{-4}$	34.54	.005269, 809757
(6)	- 6	.13564	$.404162 \times 10^{-5}$	34.54	.005269, 809721
(7)	- 6	.13564	$.404162 \times 10^{-6}$	34.54	.005269, 809717, 718737
(8)	- 6	.13564	$.404162 \times 10^{-7}$	34.54	.005269, 809717, 362305
(9)	- 6	.13564	$.404162 \times 10^{-8}$	34.54	.005269, 809717, 326662
(10)	- 6	.13564	$.404162 \times 10^{-9}$	34.54	.005269, 809717, 323099
(11)	- 6	.13564	$.404162 \times 10^{-10}$	34.54	.005269, 809717, 322743
(12)	- 6	.13564	$.404162 \times 10^{-11}$	34.54	.005269, 809717, 322706
(13)	- 6	.13564	$.404162 \times 10^{-12}$	34.54	.005269, 809717, 322702
(14)	- 6	.13564	$.404162 \times 10^{-13}$	34.54	.005269, 809717, 322702
(15)	- 6	.13564	$.404162 \times 10^{-14}$	34.54	.005269, 809717, 322702
4(16)	- 6	.13564	$.404162 \times 10^{-15}$	34.54	.005269, 809717, 322701

listing, the User's Manual and test problems are available from

(1) SHARE Library
COSMIC
University of Georgia
Athens, Georgia 30601
(Ask for Program No. 360D-13.6.007)

and from

(2) VIM Library
Software Distribution Department
Control Data Corporation
3145 Porter Drive
Palo Alto, California 94304
(Ask for Program No. G2-CAL-NLWOOD)

(d) A FORTRAN subroutine CURFIT by P. R. Bevington, listed and discussed in Reference 3.

(e) A FORTRAN program of the same title as (a) above, cited in Reference 3 and available as IBM Share Program EID-NLIN No. 3094.01.

An early version of our MARQ subroutine was based on Bevington's CURFIT[3] -- to which I am indebted -- but even our first, tentative efforts differed from CURFIT in various minor ways. (Among certain programmers — myself included — the temptation to tinker with a presented program is well-nigh irresistible. The Law of Mutual Superiority[4] applies: "Anything you have programmed, I can program better; anything I have programmed, you can program better.") Our current MARQ is barely recognizable as a descendant of CURFIT.

Subroutine MARQ has 12 arguments:

(EQS, X, Y, M, N, P, C, E, D, R, TH, EK)

Here and throughout this paper the usual convention applies to FORTRAN real and integer variable names: integer names and only integer names start with I, J, K, L, M or N. The first five arguments of MARQ are inputs:

---

3. *Philip R. Bevington, Data Reduction and Error Analysis for the Physical Sciences, McGraw-Hill, Inc., New York, 1969.*

4. *Richard V. Andree, Josephine P. Andree and David D. Andree, Computer Programming: Techniques, Analysis and Mathematics, Prentice-Hall Series in Automatic Computation, New Jersey, 1973.*

EQS = the dummy name of the subroutine that computes, for a given parameter point P, the arguments E, D and R defined below. The actual name used when calling MARQ must be declared in an EXTERNAL statement in the program that calls MARQ. (In our yaw analysis example, the actual name used is EYAW. Details of subroutine EQS/EYAW will be discussed in Section X.)

X, Y = data vectors (x_i), (y_i) where y_i is the observed value of the dependent variable y at $x = x_i$.

M = the number of data points (x_i, y_i)

= the dimension of vectors X and Y

N = the number of parameters. Note: if N exceeds 10, the dimensions of arrays ALPHA, BETA, GAMMA, PB and S on line MARQ 3 (see Appendix A) must be increased to equal or exceed N and the argument 10 on line MARQ 48 must be increased to equal the number of rows declared for GAMMA.

The next five arguments (P, C, E, D and R) serve as both input and output; the last two arguments (TH and EK) are solely outputs. Before we discuss these arguments individually, it will be helpful to see the over-all pattern as indicated in Table 3:

Table 3. Contents of MARQ Arguments P, C, E, D, R, TH and EK when Marquardt's lambda is Used

MARQ CALL NO		P	C	E	D	R	TH	EK
1	IN	P_o	- 1.	-----	-----	-----	-----	-----
	RETURN	P_o	.001A	$E(P_o)$	$D(P_o)$	$R(P_o)$	0.	-----
2	IN							
	RETURN	P_1	$\lambda(P_1)$	$E(P_1)$	$D(P_1)$	$R(P_1)$	$TH(P_1)$	$EK(P_1)$
3	IN							
	RETURN	P_2	$\lambda(P_2)$	$E(P_2)$	$D(P_2)$	$R(P_2)$	$TH(P_2)$	$EK(P_2)$
4	IN							
.								
.								
.								

From this table we note that of the last seven arguments, only P and C must contain inputs the first time MARQ is called. This first call merely obtains the error E and arrays D and R associated with the initial point P_o. Thereafter, each call returns a new, improved point; the values returned in P, C, E, D and R after the K-th call serve as input for the (K + 1)-th call. We now discuss the arguments in detail:

P = the current point P_o; that is, the current set of parameter values $\{a_j\}$ in whatever units are convenient. In each application, before MARQ is called the first time, array P must contain the initial estimated values of the N parameters in the chosen units. This same set is returned from the first call; on subsequent calls, the input P is replaced by an improved P upon return.

C = a flag (initially) and Marquardt's λ thereafter. In each application, before MARQ is called the first time, C must be set to - 2.0 or - 1.0. (The fact that C is negative alerts MARQ that it is being called for the first time.) Caution: use a FORTRAN name, not the number - 2.0 or - 1.0, in the argument list, since MARQ changes the value of this argument.

Set C = - 2.0 if the intent is to ignore the λ factor and use a straight-forward differential corrections technique. (This option is helpful, for example, if the user wants to compare the process with and without Marquardt's λ for a given set of data.) Upon return from the first call of MARQ, C will have the value zero (representing a zero λ) and will remain at zero through subsequent iterations.

Set C = - 1.0 if the intent is to use the Marquardt algorithm. Upon return from the first call of MARQ, C will have the value 0.001A (representing A times the initial value of λ). For the second and subsequent MARQ calls, the output C will be the value of Marquardt's λ that was required to obtain the improved point concurrently stored in array P. We have

$$\text{Output C} = A^r \cdot (\text{Input C})$$

where r is some integer equal to or greater than - 1 (so that the smallest possible -- and usually the most felicitous -- output C is (1/A)-th the input C).

E = a measure of the error of the fit at the point concurrently stored in array P. This measure is computed in the subroutine EQS called by MARQ and hence the coder of EQS defines this argument, preferably as the RMS or probable error given by

Eq. (20). MARQ will always return an E value less than or equal to the input E.

D = the matrix of partial derivatives, Eq. (16), evaluated at the point concurrently stored in array P:

$$D(I, K) = D_{ik} = \frac{\partial f(x, P)}{\partial a_k} \Bigg|_{\substack{x = X(I) \\ P = \text{current point}}}$$

R = the vector of residuals of the fit at the point concurrently stored in array P:

$$R(I) = Y(I) - f(x, P) \Big|_{\substack{x = X(I) \\ P = \text{current point}}}$$

TH = the angle θ, Eq. (35), in degrees. Upon return from the first call, θ is zero. Thereafter, θ is the angle between the negative gradient at the input point (not necessarily the best way to go) and the straight line from input point to output point. This angle was available within MARQ and simple curiosity prompted me to make it an output argument.

EK = the vector of estimated errors in each of the N parameters stored in array P, in the same units as the parameters. These estimates are related to output argument E by a modified form of Eq. (21) and hence if E is the RMS or probable error of the fit, then array EK contains the RMS or probable error estimates for the parameters.

In the MARQ listing, Appendix A, note that the three parameters A, B and G of Figure 2 are defined in a DATA statement, line MARQ 4 (where they are named AN, BN and GN, respectively). Thus the user can easily change the arbitrary values shown, subject to the conditions

$$A > 1, \qquad B > 1$$

and the suggested limitation

$$0 \leq G \leq 45 \text{ degrees}$$

The feature of using Eq. (37) when θ < G can be eliminated by defining GN to be zero in the DATA statement.

The data in Tables 1 and 2 were obtained with the aid of some ad hoc WRITE statements within MARQ; these statements do not appear in the listing, Appendix A.

In addition to subroutines MARQ and EQS, a matrix inversion subroutine (called on line MARQ 48) must be included in the over-all program. We have used an available subroutine MATINV listed in Appendix B. (The user may, of course, substitute the inversion scheme of his choice, modifying the CALL MATINV statement as necessary.) Upon return from MATINV, the matrix GAMMA is replaced by its inverse. MATINV contains one output statement: a SINGULAR MATRIX reprimand, which is printed only when something is very, very wrong. The integer 6 in the statement WRITE (6, 17) may have to be changed to specify the proper output unit at the user's installation.

Subroutines MARQ, MATINV, EYAW (our actual name for the dummy EQS) and EY (our program that calls MARQ) were written for use on ARDC's BRLESC I and BRLESC II computers[5]. Although the compilers in these two computers implement a FORTRAN that is not quite "standard" and not quite FORTRAN IV (and indeed not quite the same on the two computers), it is believed that the above-mentioned subroutines have been restricted for the most part to standard statements and features. One exception is the nonstandard function ARCCOS used in MARQ, line 62, to determine θ. The user can replace ARCCOS (TR) with

```
ATAN (SQRT (TR ** (- 2) - 1.0))
```

if necessary.

An important <u>omission</u> in the programs should be noted: no provision was made for double-precision arithmetic. In BRLESC I/II, all real numbers are carried with a precision of sixteen decimal digits (which is double precision on many computers) and the DOUBLE PRECISION type declaration, though permitted, acts only as a REAL declaration. For computers where double precision isn't the norm, the following DOUBLE PRECISION type declarations are recommended:

```
in MARQ   : GAMMA

in MATINV : A, T1

in EYAW   : S
```

---

5. *Lloyd W. Campbell and Glenn A. Beck, <u>BRLESC I/II FORTRAN</u>, Aberdeen Research and Development Center Technical Report No. 5, AD 704343, March, 1970.*

VIII. THE YAW EQUATION

For missiles fired in either of the two Free Flight Spark Photography Ranges[6,7] operated by the Exterior Ballistics Laboratory, we can obtain at each spark station a measurement of

(a) the down-range distance z, metres

and (b) the dimensionless complex yaw $\xi = \xi_H + i\,\xi_V$

(For our purposes — to illustrate the use of Marquardt's algorithm — definitions of the coordinate systems involved are of no interest.)

Basically, the problem is to fit a given complex yaw equation

$$\xi = f(z, P) \tag{38}$$

to a set of measurements

$$\{(z_1, \xi_{H_1}, \xi_{V_1}), (z_2, \xi_{H_2}, \xi_{V_2}), \ldots (z_m, \xi_{Hm}, \xi_{Vm})\}$$

where the z measurements are considered error-free (though of course they aren't). For most (say 99.44%) of the rounds fired in the two ranges, Equation (38) is assumed to have the form

$$\xi = K_1 e^{i\phi_1} + K_2 e^{i\phi_2} + \xi_R \tag{39}$$

where

$$K_j = K_{jo}\, e^{\lambda_j (z - z_o)} \tag{40}$$

$$\phi_j = A_j + B_j\,(z - z_o) + C_j\,(z - z_o)^2 \qquad \Big\} j = 1, 2 \tag{41}$$

ξ_R = yaw of repose

$$= -\frac{g\,(B_1 + B_2)}{B_1\, B_2\, V_o^2} \tag{42}$$

---

6. *Walter K. Rogers, Jr., "The Transonic Free Flight Range," Ballistic Research Laboratories Report No. 1044, June 1958, AD 200177.*

7. *Walter F. Braun, "The Free Flight Aerodynamics Range," Ballistic Research Laboratories Report No. 1048, July 1958, AD 202249.*

The known constants are

z_o = reference z (usually mid-range), m

V_o = velocity at z_o, m/sec

g = acceleration of gravity
= 9.80 m/sec^2 for the two ranges

and the ten unknown parameters are

$$\left.\begin{array}{ll} P(1) = \eta_1 \equiv \ell n\, K_{10} & P(6) = \eta_2 \equiv \ell n\, K_{20} \\ P(2) = \lambda_1, \text{ 1/m} & P(7) = \lambda_2, \text{ 1/m} \\ P(3) = A_1, \text{ rad} & P(8) = A_2, \text{ rad} \\ P(4) = B_1, \text{ rad/m} & P(9) = B_2, \text{ rad/m} \\ P(5) = C_1, \text{ rad/m}^2 & P(10) = C_2, \text{ rad/m}^2 \end{array}\right\} \quad (43)$$

It might seem more straightforward to define P(1) and P(6) to be the arm lengths K_{10} and K_{20}, respectively. However, pre-Marquardt experience has indicated that for our yaw problem, the differential corrections process is a little more likely to diverge when the K_{jo}'s are handled directly as parameters. For one thing, ΔK_{jo} may be so poor that (unless constraints are added), K_{jo} can go negative. This difficulty doesn't arise with η_j: any value of η_j yields a positive K_{jo}. Of course, in practice our starting point P_o is usually so well-determined that it would make no difference whether we worked with the η_j's or the K_{jo}'s.

Complex expressions and complex arithmetic are not allowed in the BRLESC I/II FORTRAN; hence we must separate Eq. (39) into its real and imaginary parts:

$$\left.\begin{array}{l} \xi_H = f_H(z, P) \equiv K_1 \cos\phi_1 + K_2 \cos\phi_2 + \xi_R \\ \xi_V = f_V(z, P) \equiv K_1 \sin\phi_1 + K_2 \sin\phi_2 \end{array}\right\} \quad (44)$$

obtaining <u>two</u> fitting equations.

IX. COMMENTS ON HANDLING TWO FITTING EQUATIONS

Previously in this report we have been considering a single fitting equation, Eq. (1). To handle two fitting equations, as in (44), we proceed as follows. Suppose that NL is the number of spark stations at which measurements were taken. Then the value of the error function, Eq. (4), is obtained by the relation

$$\varepsilon = \sum_{i=1}^{NL} \left\{ [\xi_{Hi} - f_H (z_i, P)]^2 + [\xi_{Vi} - f_V (z_i, P)]^2 \right\} \qquad (45)$$

Since MARQ has no provision for inputting more than one dependent variable, we obtain the equivalent of Eq. (45) by

(a) setting M, the number of measurements, at twice NL,

and (b) defining

$$\left.\begin{aligned} X (I) &= z_i - z_o \\ Y (I) &= \xi_{Hi} \\ Y (I + NL) &= \xi_{Vi} \end{aligned}\right\} \quad i = 1, 2, \ldots NL$$

There is no need to define the second half of vector X's M components, since it would only duplicate the first half. (Note that we need a minimum of six spark stations in order for M, the number of measurements, to exceed ten, the number of parameters.) Equation (45) then becomes

$$\varepsilon = \sum_{I=1}^{NL} [R (I)^2 + R (I + NL)^2] \qquad (46)$$

where the M residuals are obtained by the relations

$$\left.\begin{aligned} R (I) &= Y (I) - f_H (X (I), P) \\ R (I + NL) &= Y (I + NL) - f_V (X (I), P) \end{aligned}\right\} \quad I = 1, 2, \ldots NL \qquad (47)$$

The only remaining change required to handle two fitting equations instead of one, occurs in the formation of the partial derivative matrix D. Since there are two equations and ten parameters, twenty partial derivatives must be defined, ten of the form

$$D (I, K) = \frac{\partial f_H (X (I), P)}{\partial P (K)} \qquad (48)$$

and ten of the form

$$D(I + NL, K) = \frac{\partial f_V (X (I), P)}{\partial P (K)} \qquad (49)$$

To summarize, the changes needed to handle two fitting equations occur in:

(a) the manner of defining M, X and Y in the program that calls MARQ

and (b) the manner of defining E, R and D in the subroutine EQS discussed in the next section.

The technique for dealing with two fitting equations can easily be extended to any number of simultaneous fitting equations. Note that in our example, ξ_H and ξ_V are dimensionally equal. When the dependent variables are not all of the same dimensions, a little additional work is needed to insure dimensional consistency in Eq. (46) and elsewhere.

X. COMMENTS ON SUBROUTINE EQS/EYAW

The user must code the subroutine whose dummy name is EQS. The arguments of EQS must have the form prescribed by MARQ on lines 8 and 63:

(X, Y, M, N, P, E, D, R)

These eight arguments have the same meaning for EQS as they do for subroutine MARQ except that for EQS, P is an input only and E, D and R are outputs only. The user programs EQS to obtain — for given X, Y, M, N and P inputs — the error measure E, the partial derivative matrix D and the residual vector R.

For example, the pair of yaw equations (44) led to the subroutine EYAW listed in Appendix C. The velocity V_o is passed into EYAW and the yaw of repose ξ_R is extracted from EYAW by a labelled COMMON statement linked to the subroutine calling MARQ.

The definitions of the D elements within EYAW follow immediately from Equations (44), (48) and (49) - - - with one exception: I have ignored the partial derivatives of the yaw of repose with respect to the only two parameters involved therein, B_1 and B_2. Such liberties may often be taken when forming matrix D because it is <u>not</u> necessary that D be the mathematically correct set of partial derivatives. It is possible to get the right answer using a "wrong" D array, just as

we can get the right answer using a "wrong" expression for grad ϵ, Eq. (7). Of course, this doesn't mean that we can make a blunder in coding D in subroutine EQS or that we can write down any old approximation that comes to mind. A certain amount of discretion is called for; if the user has any doubts as to the merits of an approximation in D (and even if he hasn't any doubts), his safest course is to avoid such an approximation.

Subroutine MARQ receives matrix D from EQS and forms the needed elements of array α by summation, in accordance with Eq. (18). Because of the symmetry of α, only the $N \cdot (N + 1)/2$ elements on and below the principal diagonal are computed in MARQ. For our yaw equations, N is ten and thus fifty-five α elements are formed in MARQ. Note from the D equations in the EYAW listing, Appendix C, that twelve of these fifty-five elements should be identically zero. For example,

$$\alpha_{13} = \sum_{I=1}^{NL} [D(I, 1) \cdot D(I, 3) + D(J, 1) \cdot D(J, 3)]$$

where J = I + NL. Hence, in the EYAW notation,

$$\alpha_{13} = \sum [R1 \cdot (-R3) + R3 \cdot R1] = 0$$

Similarly, the reader may verify that except for sign, only nineteen of the forty-three nonzero α elements are distinct. For example,

$$\alpha_{62} = \alpha_{71} = \alpha_{84} = \alpha_{93}$$

Subroutine MARQ, of course, forms all fifty-five elements by summation; short cuts that depend on the particular fitting equation(s) used are sacrificed on the altar of generality.

XI. COMMENTS ON THE PROGRAM THAT CALLS MARQ

In the program that calls MARQ:

(a) the actual subroutine name (EYAW in our example) that will be passed as an argument to MARQ must be declared in an EXTERNAL statement.

(b) the six array arguments of MARQ must be declared in a DIMENSION statement:

X (M), Y (M), P (N), D (M, N), R (M), EK (N)

Usually, for a given problem, the value of N (the number of parameters) is known and fixed, whereas the value of M (the number of data points) varies from case to case. In that event, some number equal to or larger than the largest anticipated value of M should be used in the DIMENSION statement. (In our EBL range set-up, data can be obtained at no more than 54 spark stations; hence, we have M = 108.)

(c) each known constant in the fitting equation whose value may change from case to case (such as the reference velocity in our yaw problem) is assigned a FORTRAN name and passed to the EQS subroutine by a labelled COMMON statement. Similarly, but more rarely, any constant of interest evaluated within the EQS subroutine (such as the yaw of repose) may be rescued from oblivion by linking it through the same labelled COMMON with the program that calls MARQ.

(d) MARQ will be called repeatedly (say, in a DO-loop) until some specified convergence criterion is satisfied or until a specified number of calls have been made. For example, we might have

```
      DO 4 K = 1,KMAX
         CALL MARQ(EQS,X,Y,M,N,P,C,ENEW,D,R,TH,EK)
         IF(K.EQ.1) GOTO 3
            CR = 1.0 - ENEW/EOLD
            IF(CR.GE.0..AND.CR.LT.EPS) GOTO 5
    3    EOLD = ENEW
    4 CONTINUE
```

where the value of KMAX (the terminal parameter of the DO-loop) and the value of EPS (where 0 < EPS < < 1) have been specified before entering the DO-loop.

The process is assumed to have converged when the IF-conditions on CR above are satisfied, that is, when the ratio of the present error of the fit (ENEW) to the previous error of the fit (EOLD) falls between 1-EPS and 1. Note that the first call of MARQ, K = 1, must be handled a little differently from subsequent calls in the DO-loop because there is no zero-th error value to compare with the error returned from that first call.

Since the use of Marquardt's λ insures that ENEW $\leq$ EOLD, it may seem that CR will always be non-negative and hence that the first of the two IF-conditions is unnecessary. The catch is: whenever we avoid λ (by setting C = - 2.0), we lose our guarantee that things will improve and CR can very well go negative.

If the convergence criterion in the above DO-loop is satisfied, control is transferred to statement 5. If (KMAX - 1) iterations have

been performed without satisfying the convergence criterion, then the statement following statement 4 is executed next. What happens at these two locations is, of course, the user's affair.

For our yaw problem, the program that calls MARQ was itself written as a subroutine: the subroutine EY listed in Appendix D. The input arguments are

RD = a number identifying both the range (Aerodynamics or Transonic) and the round

N = NL, the number of spark stations

NS = an array of identifying station numbers

B = the array: measured ξ_H at each station

A = the array: measured ξ_V at each station

Z = the array: measured z at each station, m

ZR = z_o, the reference z, m

VR = velocity at z_o, m/sec

The only input/output argument is

P = the array of ten parameters defined in Eq. (43), with this exception: here P(1) and P(6) are K_{10} and K_{20}, respectively, not $\eta_j = \ln K_{jo}$ (required conversions to and from η_1 and η_2 are done within EY, so that the EY user need not be aware of the η's). Upon input to EY, P must contain the initial estimates of the parameters; upon output, P contains the final parameter values.

The output arguments of EY are:

Q = the array of estimated errors in the ten parameters. Subroutine MARQ returns to EY the estimated errors in η_1 and η_2; the errors in K_{10} and K_{20} are then obtained in EY by the approximation

$$E(K_{jo}) = K_{jo} \cdot E(\eta_j)$$

BB = the array: computed ξ_H at each station

AA = the array: computed ξ_V at each station

RMS = the root-mean-square error of the fit (Eq. (20) with b = 1.0)

YREP = the yaw of repose, Eq. (42)

IC = a convergence flag:

IC = 0 if the process has converged

= 1 if the process has failed to converge in the specified number of iterations

= 2 if the process has not been used, because there were too few (less than six) spark stations

The skeletal FORTRAN DO-loop given earlier in this section is fleshed out in EY (lines EY 41 through EY 57) by a number of relatively unimportant statements concerned with monitoring the progress of the convergence. One point concerning these statements is of minor interest. Although all angles in array P (that is, in the parameters A_1, B_1, C_1, A_2, B_2 and C_2) are in radians, the WRITE statement (line EY 52) prints the results in degrees; this is a concession to the majority of people who "can't picture radians." Someone might ask: then why not simply carry all angles within P in degrees, thereby avoiding the need to convert before printing. The answer is: conversion can't be avoided (unless we are willing to print results in radians). If the angles were carried in degrees, the conversion that we eliminated from EY would crop up in EYAW, in taking the derivatives of sines and cosines. This would be less efficient, since EYAW statements are encountered more often in the program than EY statements.

Two examples of the print-out furnished by EY (see Figures 3 and 4) are discussed in the next section.

XII. A SAMPLE CASE, WITH AND WITHOUT $\underline{\lambda}$

We define $\underline{\lambda}$ to mean "λ plus the other features in Figure 2 that distinguish Marquardt's algorithm from the usual differential corrections process." If we apply the differential corrections process with and without $\underline{\lambda}$ to the same fitting equation(s), the same set of measurements and the same starting point P_o, we can distinguish three outcomes. We will have:

(a) convergence to the same point, with or without $\underline{\lambda}$,

or (b) divergence without λ, convergence with λ,

or (c) convergence to a wrong point without λ and to a better (not necessarily the best) point with λ.

(A fourth possibility — convergence to a better point without λ than with it — is too unpleasant and, I would hope, too rare to consider here.)

Note that when λ is used, divergence is impossible: the process can never blow up. If the terminal parameter of the DO-loop is large enough, that is, if MARQ is called enough times, the process should converge to a point. (We need a limit on the number of iterations to get past the occasional pathological case.) Of course, with or without λ, "convergence" in the above triad means only that the process has stopped at some answer; we may still be miles from the right answer.

For our yaw problem, result (a) above — the least interesting result — has occurred by far the most often. For most of the rounds fired in the two EBL ranges, the motion can be so well represented by Eq. (44) and initial estimates of the ten parameters are so well determined (by a preliminary subroutine which we won't discuss here) that quick convergence is assured and Marquardt's algorithm is unnecessary.

However, I wanted to complete this report with an example of outcome (b) or (c), preferably (c) because I think it is more instructive. I could, of course, have worked with one of those relatively rare rounds for which Eq. (44) seems to be inadequate, but in that event, convergence is not necessarily an advantage. The final point reached may be the best possible based on Eq. (44) and yet be so worthless that an unwary use of the results could do more harm than no results at all.

To obtain outcome (b) or (c) from any "normal" round, I had to resort to an artifice: I by-passed the subroutine that would have given us good first estimates of the ten parameters and fed mediocre (not really bad) estimates into subroutine EY through array P. In effect, I said: let's see what happens if we get a little sloppy in choosing our starting point. For round 1-11461, the answer — outcome (c) — is illustrated in Figures 3 and 4.

Figure 3 is a print-out of the fit obtained without Marquardt's algorithm. The line TRY = 0 gives the initial estimates of the ten parameters; these produce an RMS error of 0.041732. The first iteration (TRY = 1) improves the situation but the second iteration grossly overshoots the target. It is surprising (to me, at least) that the differential corrections process could recover from such a flagrant mis-fit, but it does, converging after twenty iterations to a point P_{20} whose RMS error is 18% of the original error.

ROUND 1.11461

I* = 48.7680 M
V* = 408.868 M/SEC

TRY	K1	LAM 1 (1/M)	A1 (DEG)	B1 (DEG/M)	C1 (DEG/M**2)	K2	LAM 2 (1/M)	A2 (DEG)	B2 (DEG/M)	C2 (DEG/M**2)	MARQ LAMBDA	MARQ (DEG)	RMS ERROR
0	.01600	.00000	200.0	65.00	.000000	.06600	-.00000	150.0	20.00	.000000	0.0E 00	.00	.041732
1	.00410	.01878	240.2	63.34	-.001829	.06141	-.00368	173.6	19.15	-.015403	0.0E 00	40.73	.022777
2	1.29366	-.18722	333.1	63.52	-.063852	.06823	-.00453	169.7	19.13	-.015550	0.0E 00	34.881307	6.689284
3	.47608	-.18721	333.0	63.51	-.063809	.06962	-.00427	177.0	19.23	-.022623	0.0E 00	46.07	481.068944
4	.17520	-.18720	332.9	63.50	-.063189	.06995	-.00434	177.0	19.23	-.022511	0.0E 00	46.06	176.972275
5	.06454	-.18717	332.4	63.48	-.063468	.06995	-.00434	177.0	19.23	-.022518	0.0E 00	46.03	65.101336
6	.02383	-.18709	331.2	63.42	-.066197	.06995	-.00434	177.0	19.23	-.022517	0.0E 00	45.98	23.966319
7	.00885	-.18667	327.8	63.26	-.066198	.06995	-.00434	177.0	19.23	-.022515	0.0E 00	46.18	8.806237
8	.00335	-.18626	318.8	62.81	-.071535	.06994	-.00434	177.0	19.23	-.022510	0.0E 00	48.83	3.236523
9	.00132	-.18461	295.1	61.66	-.085523	.06993	-.00432	177.0	19.23	-.022496	0.0E 00	60.39	1.187566
10	.00059	-.18013	237.9	58.85	-.119369	.06990	-.00427	177.0	19.22	-.022452	0.0E 00	75.04	.433873
11	.00037	-.16814	132.7	53.69	-.181693	.06985	-.00411	177.0	19.22	-.022273	0.0E 00	81.76	.156830
12	.00053	-.13684	45.3	49.36	-.232458	.06977	-.00382	176.7	19.21	-.021730	0.0E 00	80.96	.055556
13	.00336	-.07514	-25.1	46.04	-.265765	.07003	-.00378	176.3	19.20	-.021005	0.0E 00	82.55	.019638
14	.01787	-.02353	-122.6	40.89	-.323578	.07134	-.00348	171.7	19.16	-.017372	0.0E 00	85.60	.011552
15	.01890	-.03458	-101.5	40.82	-.332316	.06945	-.00357	178.3	19.15	-.015969	0.0E 00	67.20	.008449
16	.01225	-.03181	-83.5	42.24	-.307995	.07069	-.00299	169.4	19.11	-.016456	0.0E 00	60.11	.008213
17	.01389	-.02862	-72.1	43.73	-.278291	.07107	-.00321	168.8	19.23	-.013051	0.0E 00	76.96	.007776
18	.01413	-.02822	-74.3	43.54	-.281404	.07013	-.00372	169.1	19.20	-.014281	0.0E 00	52.45	.007549
19	.01429	-.02793	-74.5	43.51	-.282143	.07020	-.00367	169.0	19.20	-.014208	0.0E 00	71.36	.007548
20	.01429	-.02791	-74.6	43.50	-.282259	.07021	-.00367	169.0	19.20	-.014201	0.0E 00	53.70	.007548
ERR	.00245	.00461	8.6	.49	.611856	.00165	.00079	2.3	.05	.002157	YAW OF REPOSE =		-.00825

STA	Z(M)	XI(M)	COMP	RESID	XI(V)	COMP	RESID	DELTA SA	COMP	RESID
15	4.6877	.1320	.1241	.0079	.0072	-.0021	.0093	.017448	.015393	.002075
20	6.0647	.9718	.0732	-.0014	.0904	.1014	-.0028	.014875	.015641	-.000766
22	6.6839	.0257	.0261	-.0004	.1075	.1098	-.0024	.012226	.012768	-.000542
25	7.4637	-.0227	-.0255	.0028	.0776	.0789	-.0013	.006536	.006867	-.000331
27	8.1641	-.0326	-.0317	-.0009	.0579	.0566	.0013	.004415	.004212	.000202
30	9.1951	-.0178	-.0152	-.0026	.0364	.0354	.0010	.001639	.001484	.000156
35	10.7021	-.0135	-.0212	.0077	.0599	.0608	-.0009	.063770	.005188	-.001419
40	12.2287	-.0640	-.0402	.0122	.0583	.0505	.0078	.010789	.012196	-.001406
75	22.9125	.0002	.0008	-.0006	.0558	.0536	.0023	.010904	.010759	.000145
88	26.6398	.0168	.0237	-.0069	.0000	.0799	-.0001	.006677	.006943	-.000266
90	27.4983	-.0007	-.0139	.0032	.0560	.0501	.0060	.003217	.002700	.000517
95	28.9657	-.0532	-.0408	-.0132	.0664	.0588	.0076	.007238	.005056	.002182
150	45.7565	-.0271	-.0395	.0124	.0793	.0735	.0058	.007022	.006969	.000053
155	47.2514	-.0734	-.0658	-.0075	.0408	.0362	.0046	.007048	.005643	.001406
157	47.8187	-.0814	-.0662	-.0132	.0193	.0214	-.0022	.006992	.005108	.001885
160	48.7995	-.0680	-.0652	-.0029	-.0088	-.0010	-.0078	.004708	.004248	.000460
162	49.3208	-.0576	-.0614	.0038	-.0172	-.0184	-.0049	.003615	.003876	-.000262
165	50.3428	-.0483	-.0526	.0043	-.0214	-.0245	.0031	.002787	.003369	-.000582
170	51.8737	-.0562	-.0393	-.0109	-.0308	-.0408	.0021	.004028	.003214	.000814
210	64.0582	-.0211	-.0198	-.0013	.0543	.0680	-.0138	.003391	.005018	-.001627
215	65.5789	-.0323	-.0477	.0154	.0417	.0469	-.0052	.002782	.004475	-.001694
230	70.0982	-.0535	-.0475	-.0061	-.0474	-.0337	-.0137	.005116	.003390	.001727
280	85.3934	-.0420	-.0395	-.0025	.0413	.0493	-.0075	.003514	.003998	-.000484
285	86.8806	-.0524	-.0579	.0055	.0256	.0259	-.0002	.003402	.004019	-.000617
290	88.4198	-.0663	-.0632	-.0031	.0037	-.0044	.0081	.004409	.004014	.000395
295	89.9488	-.0591	-.0537	-.0054	-.0343	-.0333	-.0010	.004666	.003986	.000680
300	91.5369	-.0271	-.0388	.0038	-.0549	-.0547	-.0002	.003744	.003938	-.000194

Figure 3. Yaw fit for Round 1-11461 from a contrived starting point, without λ.

ROUND 1.11461 Ze = 48.7680 M
Ve = 408.868 M/SEC

ITR	K1	LAM 1 (1/M)	A1 (DEG)	B1 (DEG/M)	C1 (DEG/M**2)	K2	LAM 2 (1/M)	A2 (DEG)	B2 (DEG/M)	C2 (DEG/M**2)	MARQ LAMBDA	MARQ (DEG)	RMS ERROR
0	0.01600	0.00000	200.0	65.00	0.000000	0.06600	0.00000	150.0	20.00	0.000000	0.1E-01	0.00	0.041732
1	0.01411	0.01876	260.1	63.34	-0.000937	0.06141	-0.00366	173.5	19.15	-0.015314	0.1E-02	40.64	0.022778
2	0.00730	-0.00180	267.4	63.36	-0.007106	0.06206	-0.00369	173.1	19.15	-0.015360	0.1E-03	36.82	0.019634
3	0.00829	-0.00991	268.3	63.32	-0.007181	0.06270	-0.00369	173.1	19.15	-0.015620	0.1E-04	15.58	0.017479
4	0.00880	-0.01508	268.2	63.32	-0.006605	0.06329	-0.00368	173.1	19.15	-0.015939	0.1E-05	24.86	0.015738
5	0.00915	-0.01871	267.7	63.34	-0.005542	0.06381	-0.00367	173.2	19.15	-0.016248	0.1E-06	30.48	0.014279
6	0.00943	-0.02139	267.2	63.37	-0.004131	0.06429	-0.00366	173.3	19.16	-0.016536	0.1E-07	35.48	0.013033
7	0.01217	-0.04171	262.6	63.86	0.012625	0.06876	-0.00359	173.9	19.18	-0.019159	0.1E-08	41.48	0.010405
8	0.01347	-0.03185	277.1	65.57	0.045402	0.06932	-0.00367	173.7	19.18	-0.019269	0.1E-09	81.58	0.003328
9	0.01700	-0.02459	279.6	66.32	0.062307	0.07006	-0.00348	173.8	19.17	-0.019689	0.1E-10	72.71	0.001622
10	0.01706	-0.02440	277.5	66.09	0.058064	0.07007	-0.00345	173.7	19.17	-0.019492	0.1E-11	52.45	0.001630
11	0.01714	-0.02426	277.6	66.10	0.058157	0.07008	-0.00345	173.7	19.17	-0.019473	0.1E-12	43.46	0.001629
12	0.01715	-0.02425	277.6	66.10	0.058156	0.07008	-0.00345	173.7	19.17	-0.019473	0.1E-13	44.96	0.001629
ERR	0.00063	0.00370	1.5	0.07	0.001774	0.00034	0.00316	0.4	0.01	0.008369	TAN OF REPOSE = -0.00023		

STA	Z(M)	XI(M)	COMP	RESID	XI(V)	COMP	RESID	DELTA SQ	COMP	RESID
15	6.6077	0.1320	0.1298	0.0021	0.0072	0.0063	0.0009	0.017468	0.016898	0.000570
20	6.0647	0.0718	0.0698	0.0019	0.0986	0.0996	-0.0010	0.016875	0.016791	0.000084
22	6.6639	0.0257	0.0260	-0.0003	0.1075	0.1061	0.0015	0.012226	0.011925	0.000301
25	7.6637	-0.0227	-0.0231	0.0003	0.0776	0.0781	-0.0005	0.006536	0.006636	-0.000100
27	8.1641	-0.0326	-0.0309	-0.0018	0.0579	0.0574	0.0005	0.004415	0.004248	0.000166
30	9.1951	-0.0178	-0.0184	0.0006	0.0364	0.0330	0.0034	0.001639	0.001425	0.000215
35	10.7021	-0.0135	-0.0147	0.0012	0.0599	0.0599	0.0000	0.003770	0.003802	-0.000033
40	12.2287	-0.0860	-0.0850	-0.0010	0.0583	0.0583	-0.0001	0.010789	0.010628	0.000161
75	22.9125	0.0882	0.0908	-0.0026	0.0558	0.0562	-0.0003	0.010984	0.011487	-0.000503
80	24.6390	0.0168	0.0175	-0.0007	0.0800	0.0813	-0.0013	0.006677	0.006919	-0.000242
90	27.6903	-0.0087	-0.0080	-0.0007	0.0560	0.0562	-0.0001	0.003217	0.003217	-0.000000
95	28.9657	-0.0532	-0.0540	0.0007	0.0664	0.0677	-0.0013	0.007234	0.007496	-0.000258
150	45.7595	-0.0271	-0.0276	0.0005	0.0793	0.0819	-0.0026	0.007822	0.007464	-0.000442
155	47.2514	-0.0734	-0.0754	0.0020	0.0408	0.0416	-0.0008	0.007848	0.007415	-0.000367
157	47.8107	-0.0814	-0.0786	-0.0028	0.0193	0.0195	-0.0002	0.006992	0.006554	0.000439
160	48.7995	-0.0680	-0.0671	-0.0010	-0.0088	-0.0099	0.0011	0.004708	0.004594	0.000114
162	49.3200	-0.0576	-0.0582	0.0006	-0.0172	-0.0173	0.0001	0.003415	0.003607	-0.000672
165	50.3420	-0.0483	-0.0487	0.0004	-0.0214	-0.0220	0.0006	0.002787	0.002853	-0.000066
170	51.8737	-0.0502	-0.0507	0.0005	-0.0388	-0.0421	0.0033	0.004824	0.004345	-0.000318
210	64.0582	-0.0211	-0.0230	0.0019	0.0543	0.0571	-0.0028	0.003391	0.003789	-0.000398
215	65.5789	-0.0323	-0.0337	0.0014	0.0417	0.0439	-0.0022	0.002782	0.003061	-0.000279
230	70.0962	-0.0535	-0.0538	0.0002	-0.0474	-0.0462	-0.0012	0.005116	0.005027	0.000089
280	85.3934	-0.0420	-0.0411	-0.0009	0.0418	0.0408	0.0010	0.003514	0.003358	0.000157
285	86.8806	-0.0524	-0.0495	-0.0029	0.0256	0.0253	0.0003	0.003482	0.003090	0.000392
290	88.4198	-0.0663	-0.0635	-0.0028	0.0037	0.0030	0.0007	0.004469	0.004063	0.000366
295	89.9488	-0.0591	-0.0579	-0.0012	-0.0343	-0.0342	-0.0001	0.004666	0.004523	0.000143
300	91.5369	-0.0271	-0.0272	0.0001	-0.0549	-0.0541	-0.0008	0.003744	0.003663	0.000081

Figure 4. Yaw fit for the same data and starting point as in Figure 3, but with $\underline{\lambda}$.

The point P_{20} is not at the absolute minimum but this fact can hardly be deduced from a study of Figure 3. Note the insidious lure of those columns of numbers converging to a wrong answer. If we had started with, say, P_{14} as our initial point, even our experienced analysts — who would suspect any final result that differs very much from the starting point — might be lulled into acceptance. The error estimates for the ten parameters (printed below the final iteration values) are not unacceptably large; they help maintain the illusion that the parameter values themselves are acceptable. Finally, the bottom half of Figure 3, listing the measured and computed ξ_H and ξ_V, and the corresponding residuals, seems to reinforce the impression that we have fitted the data adequately.

And yet if we look at Figure 4 — the same round, but using Marquardt's algorithm — we see that a much different fit is possible, with a much smaller RMS error and a much better set of parameter error estimates. (Thus the fact that the ten error estimates in Figure 3 were of satisfactory size proved nothing. If there had been no noise, we would have converged there to some "true local" minimum. The parameter error estimates in Figure 3 are measures of how close we came to that true local minimum, not to the true absolute minimum.)

It is instructive to see just how Marquardt's algorithm got us to the right answer in Figure 4. The first iteration is nearly the same as in Figure 3 since λ is relatively small (0.001). It is in the second iteration that Marquardt's algorithm had its first big opportunity to star. It recognized the fact that a giant over-step was about to be made and so — since θ is less than 45 degress — it divided each parameter increment by ten. For example, in Figure 3, the increment in λ_1 from the first to the second iteration is

$$(\Delta\lambda_1)_3 = -0.18722 - 0.01878 = -0.20600 \text{ (1/m)}$$

while in Figure 4, the increment is about one-tenth as large:

$$(\Delta\lambda_1)_4 = -0.00180 - 0.01876 = -0.02056 \text{ (1/m)}$$

(For K_{10} and K_{20}, recall that it is the increments in η_1 and η_2 that are reduced by a factor of ten.) A single shrinking of the step-size was sufficient in this instance to give a smaller RMS error than obtained from the first iteration and so the second iteration was concluded.

Thereafter the Marquardt part of the process had little to do. Usually, it is not possible to tell when the "$\theta < 45°$" feature has been called into play (without inserting monitoring statements within MARQ). That feature may or may not have been used whenever the print-

out shows a θ less than 45 degrees. We can say, however, that λ itself was of negligible aid in the convergence of Figure 4. This is shown by the steadily decreasing values of λ printed out. A departure from the norm

$$\lambda_{TRY\ K} = 0.1\ \lambda_{TRY\ K-1}$$

would have indicated that λ played a significant role in completing the K-th iteration. Such a departure doesn't occur in Figure 4.

Of course, if we start out anywhere within a relatively tiny region of the parameter space surrounding the solution P of Figure 4, we will converge to that solution without $\underline{\lambda}$. (This is what actually happened when we used our preliminary subroutine to determine the initial estimates.) The point is: when $\underline{\lambda}$ is used, the region of convergence is expanded considerably. Marquardt's algorithm is like radar on a ship seeking harbor; on a foggy night, we need all the help we can get.

REFERENCES

1. Donald W. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," *J. Soc. Indust. Appl. Math.*, Vol. 11, No. 2, pp 431-441, June 1963.

2. Cuthbert Daniel and Fred S. Wood, *Fitting Equations to Data: Computer Analysis of Multifactor Data for Scientists and Engineers*, Wiley-Interscience, New York, 1971.

3. Philip R. Bevington, *Data Reduction and Error Analysis for the Physical Sciences*, McGraw-Hill, Inc., New York, 1969.

4. Richard V. Andree, Josephine P. Andree and David D. Andree, *Computer Programming: Techniques, Analysis and Mathematics*, Prentice-Hall Series in Automatic Computation, New Jersey, 1973.

5. Lloyd W. Campbell and Glenn A. Beck, *BRLESC I/II FORTRAN*, Aberdeen Research and Development Center Technical Report No. 5, AD 704343, March, 1970.

6. Walter K. Rogers, Jr., "The Transonic Free Flight Range," Ballistic Research Laboratories Report No. 1044, June 1958, AD 200177.

7. Walter F. Braun, "The Free Flight Aerodynamics Range," Ballistic Research Laboratories Report No. 1048, July 1958, AD 202249.

APPENDIX A . . . SUBROUTINE MARQ

```
      SUBROUTINE MARQ(EQS,X,Y,M,N,P,C,E,D,R,TH,EK)                    ****   1
      DIMENSION X(M),Y(M),P(N),D(M,N),R(M),EK(N),                     MARQ   2
     D          ALPHA(10,10),BETA(10),GAMMA(10,10),PB(10),S(10)      MARQ   3
C
C   NOTE -- IF NO. OF PARAMETERS N.GT.10, REPLACE 10 ABOVE
C           AND IN MATINV LIST (LINE MARQ 48) BY AN INTEGER.GE.N  .
C           THE VALUES OF AN, BN AND GN BELOW MAY BE CHANGED, PROVIDED THAT
C           AN .GT. 1., BN .GT. 1. AND 0. .LE. GN .LE. 45.
C
      DATA DEG/57.29578/ CMIN/.5E-16/ AN/10./ BN/10./ GN/45./         MARQ   4
      MM = M                                                          MARQ   5
      NN = N                                                          MARQ   6
C
C               *  GO TO 1 IF MARQ IS NOT BEING CALLED FOR THE FIRST TIME.
C
      IF(C.GE.0.)GOTO 1                                               MARQ   7
C
C               *  THE FIRST TIME MARQ IS CALLED, EVALUATE E, D AND R,
C               *  SET C = INITIAL VALUE OF LAMBDA AND RETURN.
C
         CALL EQS(X,Y,MM,NN,P,E,D,R)                                  MARQ   8
         C  = 0.001*AN*(C + 2.)                                       MARQ   9
         TH = 0.                                                      MARQ  10
         RETURN                                                       MARQ  11
C
C               *  SET CL = INPUT LAMBDA/AN AND EA = INPUT ERROR.
C
    1 CL = C                                                          MARQ  12
      IF (CL.GT.CMIN) CL = CL/AN                                      MARQ  13
      EA = E                                                          MARQ  14
C
C               *  FORM THE BETA VECTOR, EQ(15), AND ONLY THE N(N+1)/2
C               *  ALPHA ELEMENTS ON OR BELOW THE PRINCIPAL DIAGONAL, EQ(18).
C
      DO 4 J = 1,NN                                                   MARQ  15
         B = 0.                                                       MARQ  16
         DO 3 K = 1,J                                                 MARQ  17
            A = 0.                                                    MARQ  18
            DO 2 I = 1,MM                                             MARQ  19
               IF(K.EQ.1) B = B + R(I)*D(I,J)                         MARQ  20
               A = A + D(I,J)*D(I,K)                                  MARQ  21
    2       CONTINUE                                                  MARQ  22
            ALPHA(J,K) = A                                            MARQ  23
    3    CONTINUE                                                     MARQ  24
         BETA(J) = B                                                  MARQ  25
    4 CONTINUE                                                        MARQ  26
```

Preceding page blank

```
C
C                *  FORM SCALE FACTORS S(J).  REPLACE BETA WITH SCALED
C                *  BETA, EQ(25).  FORM SCALED ALPHA(J,K), EQ(25), AND
C                *  STORE ABOVE THE PRINCIPAL DIAGONAL AS ALPHA(K,J).
C                *  FORM BM = THE SQUARE OF THE MAGNITUDE OF SCALED BETA.
C
      BM = 0.                                                          MARQ  27
      DO 6 J = 1,NN                                                    MARQ  28
         S(J) =1./SQRT(ALPHA(J,J))                                     MARQ  29
         BETA(J) = BETA(J)*S(J)                                        MARQ  30
         BM = BM + BETA(J)**2                                          MARQ  31
         K = J - 1                                                     MARQ  32
    5    IF(K.EQ.0)GOTO 6                                             MARQ  33
            ALPHA(K,J) = ALPHA(J,K)*S(J)*S(K)                          MARQ  34
            K = K - 1                                                  MARQ  35
            GOTO 5                                                     MARQ  36
    6 CONTINUE                                                         MARQ  37
C
C                *  FORM MATRIX GAMMA, EQ(33), BASED ON CURRENT VALUE OF LAMBDA.
C
    7 DIAG = 1. + CL                                                   MARQ  38
      DO 9 J = 1,NN                                                    MARQ  39
         GAMMA(J,J) = DIAG                                             MARQ  40
         K = J -1                                                      MARQ  41
    8    IF(K.EQ.0)GOTO 9                                             MARQ  42
            GAMMA(J,K) = ALPHA(K,J)                                    MARQ  43
            GAMMA(K,J) = GAMMA(J,K)                                    MARQ  44
            K = K - 1                                                  MARQ  45
            GOTO 8                                                     MARQ  46
    9 CONTINUE                                                         MARQ  47
C
C                *  REPLACE GAMMA BY ITS INVERSE.
C
      CALL MATINV(GAMMA,NN,PB,10,0,DOT)                                MARQ  48
C
C                *  FORM THE COMPONENTS OF THE SCALED DELTA P VECTOR...
C                *        DP = SCALED (DELTA A) SUB J, SATISFYING EQ(32).
C                *  FORM THE CANDIDATE POINT PB = P + UNSCALED DELTA P.
C                *  FORM DOT = DOT PRODUCT OF SCALED BETA AND SCALED DELTA P.
C                *  FORM DPM = THE SQUARE OF THE MAGNITUDE OF THE SCALED DELTA P.
C
      DOT = 0.                                                         MARQ  49
      DPM = 0.                                                         MARQ  50
      DO 11 J = 1,NN                                                   MARQ  51
         DP = 0.                                                       MARQ  52
         DO 10 K = 1,NN                                                MARQ  53
            DP = DP + BETA(K)*GAMMA(J,K)                               MARQ  54
   10    CONTINUE                                                      MARQ  55
         PB(J) = P(J) + DP*S(J)                                        MARQ  56
         DOT = DOT + DP*BETA(J)                                        MARQ  57
         DPM = DPM + DP*DP                                             MARQ  58
   11 CONTINUE                                                         MARQ  59
```

```
C
C                 *  FORM ANG = THETA, DEGREES, EQ(35).
C
      ANG = 0.                                                          MARQ  60
      TR = DOT/SQRT(DPM*BM)                                             MARQ  61
      IF(ABS(TR).LE.1.) ANG = DEG*ARCCOS(TR)                            MARQ  62
C
C                 *  EVALUATE EB = ERROR AT THE CANDIDATE POINT PB.
C
   12 CALL EQS(X,Y,MM,NN,PB,EB,D,R)                                     MARQ  63
C
C                 *  COMPARE ERROR EB AT POINT PB WITH INPUT ERROR EA.
C
      IF(EB.LE.EA)GOTO 15                                               MARQ  64
      IF(CL.EQ.0.)GOTO 15                                               MARQ  65
         IF(ANG.LT. GN)GOTO 13                                          MARQ  66
C
C                 *  INCREASE LAMBDA AND GO BACK TO COMPUTE NEW GAMMA.
C
            CL = AN*CL                                                  MARQ  67
            GOTO 7                                                      MARQ  68
C
C                 *  DECREASE LENGTH OF DELTA P AND GO BACK TO COMPUTE NEW EB.
C
   13    DO 14 J = 1,NN                                                 MARQ  69
            PB(J) = P(J) + (PB(J)-P(J))/BN                              MARQ  70
   14    CONTINUE                                                       MARQ  71
         GOTO 12                                                        MARQ  72
C
C                 *  THE ITERATION HAS BEEN COMPLETED SATISFACTORILY.
C                 *  UPDATE CURRENT POINT P AND ERROR E.  COMPUTE ERROR
C                 *  ESTIMATES FOR THE PARAMETERS.
C
   15 E  = EB                                                           MARQ  73
      C  = CL                                                           MARQ  74
      TH = ANG                                                          MARQ  75
      DO 16 J = 1,NN                                                    MARQ  76
         P(J) = PB(J)                                                   MARQ  77
         EK(J) = EB*S(J)*SQRT( GAMMA(J,J)*DIAG )                        MARQ  78
   16 CONTINUE                                                          MARQ  79
      RETURN                                                            MARQ  80
      END                                                               MARQ  81
```

APPENDIX B . . . SUBROUTINE MATINV

OBTAINED FROM COMPUTER SUPPORT DIVISION
ABERDEEN RESEARCH AND DEVELOPMENT CENTER

```
      SUBROUTINE MATINV(A,N,C,NMAX,K,DET)                          *****  1
      DIMENSION A(NMAX,1),C(1)                                     MATINV 2
      NN = N                                                       MATINV 3
      KK = K                                                       MATINV 4
      IF (1-KK) 3,1,1                                              MATINV 5
    1 N3 = NN                                                      MATINV 6
      IF (KK) 2,4,2                                                MATINV 7
    2 ASSIGN  9 TO N5                                              MATINV 8
      ASSIGN 13 TO N7                                              MATINV 9
      GOTO 5                                                       MATINV10
    3 N3 = KK + NN - 1                                             MATINV11
    4 ASSIGN 10 TO N5                                              MATINV12
      ASSIGN 14 TO N7                                              MATINV13
    5 DET = 1.0                                                    MATINV14
      DO 15 I = 1,NN                                               MATINV15
         IF (A(I,I)) 7,6,7                                         MATINV16
    6    WRITE(6,17)                                               MATINV17
         DET = 0.0                                                 MATINV18
         GOTO 16                                                   MATINV19
    7    T1 = 1.0/A(I,I)                                           MATINV20
         DET = DET*A(I,I)                                          MATINV21
         A(I,I) = 1.0                                              MATINV22
         DO 8 J = 1,N3                                             MATINV23
            A(I,J) = A(I,J)*T1                                     MATINV24
    8    CONTINUE                                                  MATINV25
         GOTO N5, (9,10)                                           MATINV26
    9    C(I) = C(I)*T1                                            MATINV27
   10    DO 14 J = 1,NN                                            MATINV28
            IF (I-J) 11,14,11                                      MATINV29
   11       T1 = A(J,I)                                           MATINV30
            A(J,I) = 0.0                                           MATINV31
            DO 12 L = 1,N3                                         MATINV32
               A(J,L) = A(J,L) - T1*A(I,L)                         MATINV33
   12       CONTINUE                                              MATINV34
            GOTO N7, (13,14)                                       MATINV35
   13       C(J) = C(J) - T1*C(I)                                 MATINV36
   14    CONTINUE                                                  MATINV37
   15 CONTINUE                                                     MATINV38
   16 RETURN                                                       MATINV39
   17 FORMAT (16H SINGULAR MATRIX)                                 MATINV40
      END                                                          MATINV41
```

APPENDIX C . . . SUBROUTINE EYAW

```
      SUBROUTINE EYAW(X,Y,M,N,P,E,D,R)                                  ****    1
      DIMENSION X(M),Y(M),P(N),D(M,N),R(M)                              EYAW    2
      COMMON/EEP/V,F                                                    EYAW    3
      S = 0.                                                            EYAW    4
      F = 0.                                                            EYAW    5
      T = P(4)*P(9)*V*V                                                 EYAW    6
      IF(T.NE.0.) F = -9.8*( P(4) + P(9) )/T                            EYAW    7
      NL = M/2                                                          EYAW    8
      DO 2 I =1,NL                                                      EYAW    9
         J = I + NL                                                     EYAW   10
         W = X(I)                                                       EYAW   11
         E1 = EXP( P(1) + P(2)*W )                                      EYAW   12
         E2 = EXP( P(6) + P(7)*W )                                      EYAW   13
         A1 = P(3) + W*( P(4) + W*P(5) )                                EYAW   14
         A2 = P(8) + W*( P(9) + W*P(10))                                EYAW   15
         R1 = E1*COS(A1)                                                EYAW   16
         R2 = E2*COS(A2)                                                EYAW   17
         R3 = E1*SIN(A1)                                                EYAW   18
         R4 = E2*SIN(A2)                                                EYAW   19
         R(I) = Y(I) - R1 -R2 - F                                       EYAW   20
         R(J) = Y(J) - R3 -R4                                           EYAW   21
         S = S + R(I)**2 + R(J)**2                                      EYAW   22
         D(I,1) =  R1                                                   EYAW   23
         D(J,1) =  R3                                                   EYAW   24
         D(I,2) =  R1*W                                                 EYAW   25
         D(J,2) =  R3*W                                                 EYAW   26
         D(I,3) = -R3                                                   EYAW   27
         D(J,3) =  R1                                                   EYAW   28
         D(I,4) = -D(J,2)                                               EYAW   29
         D(J,4) =  D(I,2)                                               EYAW   30
         D(I,5) = -D(J,2)*W                                             EYAW   31
         D(J,5) =  D(I,2)*W                                             EYAW   32
         D(I,6) =  R2                                                   EYAW   33
         D(J,6) =  R4                                                   EYAW   34
         D(I,7) =  R2*W                                                 EYAW   35
         D(J,7) =  R4*W                                                 EYAW   36
         D(I,8) = -R4                                                   EYAW   37
         D(J,8) =  R2                                                   EYAW   38
         D(I,9) = -D(J,7)                                               EYAW   39
         D(J,9) =  D(I,7)                                               EYAW   40
         D(I,10)= -D(J,7)*W                                             EYAW   41
         D(J,10)=  D(I,7)*W                                             EYAW   42
      2 CONTINUE                                                        EYAW   43
        E = SQRT( S/FLOAT(M-10) )                                       EYAW   44
        RETURN                                                          EYAW   45
        END                                                             EYAW   46
```

```
      SUBROUTINE EY(RD,N,NS,B,A,Z,ZR,VR,P,Q,BB,AA,RMS,YREP,IC)          ****    1
      EXTERNAL EYAW                                                     EY      2
      DIMENSION NS(N),B(N),A(N),Z(N),P(10),X(108),Y(108),D(108,10),     EY      3
     1  R(108),Q(10),BB(N),AA(N)                                       EY      4
      COMMON/EEP/V,YR                                                   EY      5
      DATA DEG/57.29578/                                                EY      6
   40 FORMAT(1H0,17H TOO FEW STATIONS/)                                 EY      7
   41 FORMAT(1H1,20X,6HROUND ,F8.5,20X,5HZ* = ,F10.4,2H M/              EY      8
     1 1H ,54X,5HV* = ,F9.3,7H  M/SEC//5H  TRY,                         EY      9
     2  5X,2HK1,5X,5HLAM 1,5X,2HA1, 6X,2HB1,8X,2HC1,                   EY     10
     3  9X,2HK2,5X,5HLAM 2,5X,2HA2, 6X,2HB2,8X,2HC2,                   EY     11
     4  8X,4HMARQ,4X,4HMARQ,3X,3HRMS/                                  EY     12
     5 1H ,7X,2( 9X,33H(1/M)   (DEG)  (DEG/M) (DEG/M**2),4X),          EY     13
     6 6HLAMBDA,2X,5H(DEG),5X,5HERROR/)                                 EY     14
   42 FORMAT(1H ,I4,2(F8.5,F9.5,F8.1,F8.2,F11.6,2X),E9.1,F7.2,F11.6)    EY     15
   43 FORMAT(1H0,4H ERR,2(F8.5,F9.5,F8.1,F8.2,F11.6,2X),5X,             EY     16
     1 16HYAW OF REPOSE = ,F8.5//)                                      EY     17
   44 FORMAT(43H ****WARNING**** PROCESS FAILED TO CONVERGE//)          EY     18
   45 FORMAT(5H  STA,4X,4HZ(M),9X,21HXI(M)    COMP   RESID,7X,         EY     19
     1 21HXI(V)    COMP   RESID,7X,8HDELTA SQ,5X,4HCOMP,5X,5HRESID/)    EY     20
   46 FORMAT(1H ,I4,F10.4,2(4X,3F8.4),4X,3F10.6)                        EY     21
      NL = N                                                            EY     22
      M  = NL + NL                                                      EY     23
      DF = M - 10                                                       EY     24
      IF (DF.GT.0.) GOTO 1                                              EY     25
          WRITE(6,40)                                                   EY     26
          IC = 2                                                        EY     27
          GOTO 7                                                        EY     28
    1 IC = 0                                                            EY     29
      V  = VR                                                           EY     30
      C  = -1.0                                                         EY     31
      DO 2 I = 1,NL                                                     EY     32
         J = I + NL                                                     EY     33
         X(I) = Z(I) - ZR                                               EY     34
         Y(I) = B(I)                                                    EY     35
         Y(J) = A(I)                                                    EY     36
    2 CONTINUE                                                          EY     37
      WRITE(6,41) RD,ZR,V                                               EY     38
      P(1) = ALOG( P(1) )                                               EY     39
      P(6) = ALOG( P(6) )                                               EY     40
```

```
      DO 4 K = 1,26                                                          EY     41
         CALL MARQ(EVAW,X,Y,M,10,P,C,E,D,R,TH,Q)                             EY     42
         R1 = EXP( P(1) )                                                    EY     43
         A1 = DEG*P(3)                                                       EY     44
         B1 = DEG*P(4)                                                       EY     45
         C1 = DEG*P(5)                                                       EY     46
         R2 = EXP( P(6) )                                                    EY     47
         A2 = DEG*P(8)                                                       EY     48
         B2 = DEG*P(9)                                                       EY     49
         C2 = DEG*P(10)                                                      EY     50
         J  = K - 1                                                          EY     51
         WRITE(6,42) J,R1,P(2),A1,B1,C1,R2,P(7),A2,B2,C2,C,YH,E              EY     52
         IF (J.EQ.0) GOTO 3                                                  EY     53
             CR = 1.0 - E/EA                                                 EY     54
             IF (CR .GE. 0. .AND. CR .LT. .000010) GOTO 5                    EY     55
3        EA = E                                                              EY     56
4     CONTINUE                                                             EY     57
      IC = 1                                                               EY     58
      WRITE(6,44)                                                          EY     59
5     Q3 = DEG*Q(3)                                                        EY     60
      Q4 = DEG*Q(4)                                                        EY     61
      Q5 = DEG*Q(5)                                                        EY     62
      Q8 = DEG*Q(8)                                                        EY     63
      Q9 = DEG*Q(9)                                                        EY     64
      Q10= DEG*Q(10)                                                       EY     65
      P(1) = R1                                                            EY     66
      P(6) = R2                                                            EY     67
      Q(1) = R1*Q(1)                                                       EY     68
      Q(6) = R2*Q(6)                                                       EY     69
      WRITE(6,43) Q(1),Q(2),Q3,Q4,Q5,Q(6),Q(7),Q8,Q9,Q10,YR                EY     70
      RMS = E                                                              EY     71
      YREP= YR                                                             EY     72
      WRITE(6,45)                                                          EY     73
      DO 6 I = 1,NL                                                        EY     74
         J = I + NL                                                        EY     75
         BB(I) = Y(I) - R(I)                                               EY     76
         AA(I) = Y(J) - R(J)                                               EY     77
         DA= Y(I)**2 + Y(J)**2                                             EY     78
         DB= BB(I)**2 + AA(I)**2                                           EY     79
         DC= DA - DB                                                       EY     80
         WRITE(6,46) NS(I),Z(I),Y(I),BB(I),R(I),Y(J),AA(I),R(J),DA,DB,DCEY     81
6     CONTINUE                                                             EY     82
7     RETURN                                                               EY     83
      END                                                                  EY     84
```

LIST OF SYMBOLS

A	(1) a factor greater than unity by which λ is to be increased if necessary to insure that $\varepsilon_1 < \varepsilon_o$
	(2) an EY input array argument: the measured ξ_V values at each station
A_1, A_2	orientation angles of the two yaw arms at z_o, Eq. (41); two of the ten yaw parameters
AA	an EY output array argument: the computed ξ_V values at each station
AN	the FORTRAN name for A (def. 1) in subroutine MARQ
a_1, a_2, ...a_n	the n parameters of the fitting equation whose values are to be determined
$\hat{a}_k$	$a_k \sqrt{\alpha_{kk}}$
B	(1) a factor greater than unity by which $\vec{\Delta P}$ is to be decreased if necessary when $\theta < G$
	(2) an EY input array argument: the measured ξ_H values at each station
B_1, B_2	turning rates of the two yaw arms at z_o, Eq. (41); two of the ten yaw parameters
BB	an EY output array argument: the computed ξ_H values at each station
BN	the FORTRAN name for B (def. 1) in subroutine MARQ
b	1 or 0.67449, Eq. (20)
C	a MARQ input/output argument; after the first call, C = λ
C_1, C_2	two of the ten yaw parameters, Eq. (41)
D	a MARQ input/output argument and an EQS/EYAW output argument: the array of partial derivatives D_{ik}
DET	determinant of matrix γ

D_{ik}	$\left[\frac{\partial f\ (x_i,\ P)}{\partial a_k}\right]_o$
$\hat{D}_{ik}$	$D_{ik} / \sqrt{\alpha_{kk}}$
E	estimate of the error of the fit, Eq. (20); a MARQ input/output argument and an EQS/EYAW output argument
E_j	estimate of the error in parameter a_j, Eq. (21)
EK	a MARQ output array argument: the estimated errors E_j
EQS	a MARQ input argument: the dummy name of a subroutine called by MARQ
EY	the subroutine that calls MARQ in the yaw problem
EYAW	the actual name of EQS in the yaw problem
F	the criterion function, Eq. (4)
f	the fitting function, Eq. (1)
f_H, f_V	the two fitting functions for the yaw problem, Eq. (44)
G	the value of θ (degrees) below which a new $\vec{\Delta P}$ is obtained by shrinking the current $\vec{\Delta P}$.
GN	the FORTRAN name for G in subroutine MARQ
g	the acceleration of gravity, assumed constant, Eq. (42)
h	a dimensionless positive constant, Eq. (30-31)
IC	an EY output argument: a convergence flag
K_1, K_2	lengths of the two yaw arms, Eq. (39)
K_{10}, K_{20}	values of K_1, K_2 at z_0, Eq. (40)
L	a local minimum point
M	a MARQ and EQS/EYAW input argument: the number of data points, m

MARQ	the subroutine for carrying out the Marquardt algorithm of Figure 2
MATINV	a matrix inversion subroutine
m	the number of measurements: the number of data points (x_i, y_i)
N	(1) a MARQ and EQS/EYAW input argument: the number of parameters, n (2) an EY input argument: the number of spark stations, NL
NL	the number of spark stations at which measurements were taken, M/2
NS	an EY input argument: an array of station numbers
n	the number of parameters
P	(1) the parameter set $\{a_1, a_2, \ldots a_n\}$; a point in n-dimensional parameter space (2) an EY and MARQ input/output array argument and an EQS/EYAW input array argument: the parameter values
P_g	a point along the negative gradient
$P_o, P_1, P_2, \ldots$	the initial and successive parameter points in the iterative fitting process
$\vec{P}$	the vector from the origin of the parameter space to point P
Q	an EY output argument: the array of estimated errors in the ten yaw parameters
R	a MARQ input/output array argument and an EQS/EYAW output array argument: the residuals of the fit (observed—computed)
RD	an EY input argument: a number identifying both the range and the round
RMS	an EY output argument: the RMS error of the yaw fit

r	an integer ≥ -1
S	n-dimensional parameter space, in which the coordinates of a point P are the values of the n parameters
$\hat{S}$	scaled parameter space, Eq. (23), in which each coordinate has the dimensions of the dependent variable y
TH	a MARQ output argument, the angle θ (degrees)
V_o	velocity at z_o, m/sec
VR	an EY input argument: V_o; passed to EYAW (under alias V) by labelled COMMON
X	a MARQ and EQS/EYAW input argument: the array of data points (x_i)
x	the independent variable of the fitting equation
x_i	value of x at which a y measurement was taken, $i = 1, 2, \ldots m$
Y	the set of measurements $\{y_i\}$; a MARQ and EQS/EYAW input array argument
YREP	an EY output argument: ξ_R, the yaw of repose; obtained from EYAW (under aliases YR and F) by labelled COMMON
y	the dependent variable of the fitting equation
y_i	the measured value of y at $x = x_i$, $i = 1, 2, \ldots m$
Z	an EY input argument: the array of z measurements
ZR	an EY input argument: z_o
z	down-range distance, metres
z_o	reference z value, metres
α	the curvature matrix $\left(\alpha_{jk}\right)_S$

$\tilde{\alpha}$	the inverse of matrix α
$\hat{\alpha}$	the scaled curvature matrix $\left(\hat{\alpha}_{jk}\right)_{\hat{S}}$
α_{jk}	$\frac{1}{2}\left[\frac{\partial^2 \epsilon}{\partial a_j \partial a_k}\right]_o$ Eq. (11)
$\tilde{\alpha}_{jk}$	$\frac{\text{cofactor of } \alpha_{jk}}{\text{determinant of } \alpha}$
$\hat{\alpha}_{jk}$	$\alpha_{jk} \Big/ \sqrt{\alpha_{jj}\ \alpha_{kk}}$
$\tilde{\hat{\alpha}}_{jk}$	$\frac{\text{cofactor of } \hat{\alpha}_{jk}}{\text{determinant of } \hat{\alpha}}$
$\vec{\beta}$	$-\frac{1}{2}\,(\text{grad}\ \epsilon)_o$, Eq. (13) $= \left(\beta_1, \beta_2, \ldots \beta_n\right)_S = \left(\hat{\beta}_1, \hat{\beta}_2, \ldots \hat{\beta}_n\right)_{\hat{S}}$
β_k	$-\frac{1}{2}\left[\frac{\partial \epsilon}{\partial a_k}\right]_o$, Eq. (10)
$\hat{\beta}_k$	$\beta_k \Big/ \sqrt{\alpha_{kk}}$, Eq. (25)
γ	the modified $\hat{\alpha}$ matrix $\left(\gamma_{jk}\right)_{\hat{S}}$
γ_{jk}	$\begin{cases} 1 + \lambda, & j = k \\ \hat{\alpha}_{jk}, & j \neq k \end{cases}$ Eq. (33)
Δ	a positive constant << 1 in the relaxed standard, Eq. (36b)
Δa_j	the change in a_j in moving from one point to the next in the iterative fitting process
$\Delta \hat{a}_j$	$\Delta a_j \cdot \sqrt{\alpha_{jj}}$

LIST OF SYMBOLS (continued)

$\overrightarrow{\Delta P}$	the vector $\left(\Delta a_j\right)_S = \left(\Delta \hat{a}_j\right)_{\hat{S}}$ from the current parameter point to a new point
ϵ	F(P), the sum of the squares of the residuals of the fit at point P, Eq. (4)
$\epsilon_o, \epsilon_1, \ldots$	ϵ values associated with points P_o, P_1, . . .
η_1, η_2	$\ell n\ (K_{10})$, $\ell n\ (K_{20})$
θ	the angle between $\overrightarrow{\beta}$ and $\overrightarrow{\Delta P}$, that is, between the negative gradient and the direction actually taken, Eq. (35)
λ	a dimensionless positive constant added to the diagonal elements of $\hat{\alpha}$ in order to effect an interpolation between the methods of steepest descent and differential equations.
λ_M	the smallest value of λ needed to satisfy Eq. (36)
$\underline{\lambda}$	notation for the phrase "λ plus the other features in Figure 2 that distinguish Marquardt's algorithm from the usual differential corrections process."
ξ	the complex yaw, $\xi_H + i\ \xi_V$, Eq. (39)
ξ_H, ξ_V	the components of the complex yaw, Eq. (44)
ξ_R	the yaw of repose, Eq. (42)
ϕ_1, ϕ_2	orientation angles of the two yaw arms, Eq. (39)
$[\ ]_d$	dimensions of the bracketed expression
$[\ ]_o$	evaluation at the current set of parameter values